MASTER EN SCIENCES INFORMATIQUES

Contribution à la Conception et à la Réalisation d'un SGBD d'objets bureautiques

Hubar, Pierre; Tasseroul, Daniel

Award date:
1989

Awarding institution:
Universite de Namur

Année académique 1988 - 1989.

Contribution à la Conception et à la Réalisation d'un SGBD d'objets bureautiques

Mémoire présenté en vue de l'obtention du diplôme
de Licence et Maître en Informatique.

Pierre HUBAR
Daniel Tasseroul
Promoteur : R.LESUISSE

REMERCIEMENTS

Nous tenons à remercier Monsieur Roland Lesuisse, promoteur de ce mémoire, pour l'intérêt qu'il a manifesté à l'égard de ce travail.

Nous tenons également à remercier Monsieur Jean-Luc Hainaut pour l'aide et les nombreux conseils qu'il nous a prodigués.

A tous deux, nous sommes reconnaissant d'avoir contribué à notre formation d'informaticien. Qu'ils trouvent ici l'expression de notre gratitude et de notre reconnaissance.

TABLE DES MATIERES

CHAPITRE III : L'ARCHITECTURE DES DONNEES

CHAPITRE IV : SPECIFICATIONS DES PRIMITIVES

CHAPITRE V : EXEMPLES D'APPLICATION

INTRODUCTION

A l'instar de nombreux autres secteurs, l'informatique a fait son apparition dans les bureaux où elle a progressivement touché l'ensemble des fonctions qui y sont couramment exercées. Ainsi est né le concept de **bureautique**.

Parmi la multitude d'objets manipulés dans un bureau le document occupe une place fondamentale En effet, tout document constitue la matérialisation d'informations, et en tant que support, il permet l'échange, la diffusion, la conservation,... de celles-ci.

L'importance d'un document résidant dans sa fonction de mémorisation, il est vital - afin de pouvoir prendre connaissance de l'information - de le retrouver aisément, et toute recherche de celui-ci sera d'autant plus facile si, lors de son rangement, il a été classé.

Il ressort de ceci que le rangement (incluant le classement et le stockage) est une fonction essentielle au sein d'un bureau. Aussi, plus que toutes autres, elle mérite qu'on envisage son automatisation. L'application des techniques informatiques à cette dernière peut apporter à la fois un supplément de rigueur dans le classement, et une réduction de la charge de travail.

Ce mémoire aura pour ambition de contribuer à l'automatisation de la fonction de rangement. Il consistera d'abord à isoler les concepts associés à un bureau puis à modéliser les concepts propres au rangement en suivant une approche orientée objet. Cette approche orientée objet sera abandonnée lors de l'implémentation puisque l'outil utilisé - en l'occurrence une base de donnée de type réseau - ne permet pas d'implémenter en terme du concept d'**objet**. La réalisation des types abstraits relatifs aux concepts de rangement s'appuiera sur N.D.B.S (Network Data Base System), un Système de Gestion de Bases de Données développé à l'Institut d'Informatique par M. J-L. HAINAUT et son équipe, et sur le langage de programmation PASCAL.

En pratique, les services de rangement (opérations) seront assurés par un ensemble de procédures PASCAL destinées à être utilisée sur micro-ordinateurs. Ces primitives visent donc à permettre à un programmeur d'application de manipuler des objets de bureau tels que des tiroirs, des piles,...,des documents; objets dont la description et la représentation dans la base de données lui sont cachées.

En dernier lieu, un programme d'application sera implémenté afin d'illustrer l'emploi et la puissance des primitives offertes.

Présentation de la structure du mémoire

Dans le chapitre I, nous décrivons le contexte de la bureautique et nous présentons deux recherches relatives à ce domaine et axées sur les bases de données.

Ensuite, dans le chapitre II, nous passons en revue les objets et les opérations attachés à un environnement de rangement. L'étude, réalisée avec l'appui de l'approche orientée objet, examine d'abord le rangement manuel d'un document avant d'envisager ce que pourrait être un rangement automatisé. Le chapitre se conclut par un inventaire de toutes les opérations possibles au sein de l'environnement de rangement automatisé.

Le chapitre III introduit l'outil qui est à notre disposition (N.D.B.S) et expose la réalisation des types abstraits manipulés par les primitives de rangement. Les spécifications externes de ces dernières et des commentaires sur leur implémentation font l'objet du chapitre IV.

Le chapitre V illustre, notamment par un programme d'application, l'emploi de ces primitives.

CHAPITRE I : LA BUREAUTIQUE

Dans ce chapitre, nous tentons de cerner le terme "bureautique" ainsi que les concepts l'accompagnant habituellement, c'est-à-dire le bureau et l'information. En plus d'une définition du concept de bureautique, la première section veut apporter au lecteur un ensemble de réflexions qui lui permettront de mieux percevoir ce concept. Par après -c'est l'objet de la deuxième section- nous exposons deux applications des bases de données en bureautique.

Tout au long de ces sections, nous essayons le plus fréquement possible de nous positionner par rapport à notre problème Nous délimitons ainsi précisément le domaine de notre étude et la façon dont celle-ci est menée.

I.1. LE CONTEXTE DE LA BUREAUTIQUE

Non obstant les sens classiques et divers qui sont donnés au terme **bureau**, nous le définirons comme un "local où l'on crée (par l'écriture), manipule et range de l'information dont on veut se souvenir. Cette information est l'image d'un processus de production". [LESUISSE,89]

Ainsi, une autre notion importante apparaît, à savoir celle d'**information.** D'après F. Bodart, "l'information est une signification potentielle attachée aux données; une donnée étant une représentation codée de faits relatifs à des objets, des concepts,des événements". [BODART,1988]

L'information peut se représenter sous différents modes : le mode écrit, le mode oral, le mode graphique... et, de plus en plus, on rencontre des modes de représentation mixte (écrit-graphique, graphique-son,...). Les exemples d'information étant légions, nous n'en citons ici que quelques uns :

- un message écrit ou oral (avis, entretien téléphonique,...);
- un document écrit (lettre,...);
- un bilan d'entreprise où par exemple le mode de représentation est mixte : écrit et graphique.

Donc, l'information constitue bien l'objet principalement manipulé dans un bureau. En dehors de ses divers modes de représentation, une information est toujours perçue conceptuellement comme la réunion de deux composants à savoir :

- d'une part, un **en-tête** qui regroupe l'ensemble des attributs caractérisant une information. Cet en-tête permet la manipulation de l'information (son rangement, son accès,...);
- d'autre part, un **corps** qui est le contenu proprement dit de l'information.

Dans son travail quotidien, l'employé de bureau édite des documents, remplit des formulaires, échange de l'information, la diffuse, la transforme, la duplique, la classe, archive des dossiers, organise, décide,... Or, depuis quelques années, avec l'introduction de l'informatique et de certaines technologies associées, est né le concept de **bureautique** défini comme: "...l'assistance aux travaux de bureau, procurée par des moyens et des procédures faisant appel aux techniques de l'informatique, des télécommunications et de l'organisation administrative et, de façon générale, à tout ce qui concourt à la logistique du bureau et de son environnement . Plus conceptuellement, la bureautique intéresse le système d'information individuel de toute personne travaillant dans un bureau, sans exiger d'elle d'autres connaissances que celles de son savoir-faire professionnel". [DE BLASIS,1982]

On peut ainsi dire que "la bureautique recouvre les équipements de traitement de texte, de l'image et de la parole, et fait appel aux moyens de télécommunications les plus variés. Elle vise une gestion plus efficace des documents et permet d'imaginer à terme une conception du bureau sans papier". [Simon Nora, Alain Minc ,]

Remarquons immédiatement que la bureautique, comme les techniques sur lesquelles elle s'appuie, évolue de "jour en jour". Notons également que, par l'introduction de l'informatique, un nouveau mode de représentation de l'information apparaît dans le bureau. Il s'agit du mode **"informatique"** , c'est-à-dire digital ou binaire; mode qui donne lieu à une uniformisation de la représentation d'une information. Cette dernière peut être digitalisée et ensuite rendue sous son mode de représentation initial.

Ainsi que nous l'avons précédemment signalé, la bureautique vise à faciliter l'exécution des diverses tâches exercées au sein d'un bureau. On parle de bureautique de support lorsque celle-ci autorise un travail plus confortable et plus efficace. Les domaines d'application de la bureautique sont donc divers. Mentionnons par exemple :

- l'édition de documents par le biais de traitements de texte;
- le remplissage de formulaires via une application (programme informatique);
- l'échange d'informations entre bureaux via un réseau de télécommunication, des supports de mémorisation,...;
- la diffusion d'informations par l'utilisation d'un réseau de télécommunication, d' imprimantes,...;
- la classification, l'archivage (stockage) grâce à des logiciels et équipements divers (disques, bandes magnétiques,...);
- la manipulation des données (base de données,...);
- ...

Le domaine d'application qui nous intéresse dans le cadre du mémoire, se focalise sur le **classement** et le **stockage**. Ces deux fonctions sont, dans n'importe quelle entreprise, loin d'être négligeables et aucune catégorie de personnel (cadre, employé de bureau...) n'y échappe.

Suite à l'utilisation de plus en plus fréquente de traitements de texte ou de tableurs, le personnel de bureau visualise généralement l'information, qu'il a le plus souvent créée lui-même, sous forme de données informatiques (fichiers informatiques). Ces fichiers, il doit les stocker afin de conserver l'information qu'il vient de créer. De ce fait, on peut envisager -dans un futur- que le responsable de bureau ne manipule pratiquement plus de papier - si ce n'est à des fins de difffusion.

Mais même dans un bureau "fortement" informatisé, l'employé manipule encore des informations sous une forme plus traditionnelle; informations qui ne sont ni stockées, ni exploitées via des moyens électroniques (exemple: un contrat signé, des documents volumineux tels que dictionnaires,...). Il en résulte, au sein du bureau, la coexistence d'informations sous formes diverses : fichiers informatiques, livres, bandes magnétiques, ...

Dans un environnement de rangement manuel, la **classification** des informations constitue une activité importante car chaque responsable maintient des fichiers -au sens classique du terme- d'une manière ou d'une autre. Ce processus de classification se trouve "caractérisé par une organisation arborescente , une personnalisation très marquée (choix de l'organisation et caractère privé ou confidentiel des informations) et des réorganisations fréquentes"[SOULIER,1984] .Un des buts de ce mémoire consiste à automatiser ce processus de classement et d'accès au contenu des fichiers (sens collection de fiches). Cette volonté se trouve renforcée par le fait que l'on ne manipulera pratiquement plus de papier mais bien des fichiers informatiques et que l'on apportera une rigueur dans le classement.

Le classement automatisé reprend le principe actuellement mis en oeuvre qui est fondé sur "l'observation que la plupart des gens, dans un environnement de rangement manuel, sont capables de retrouver, assez vite, le contenu de leurs fichiers (au sens classique) seulement en jetant un rapide coup d'oeil sur les onglets de leurs dossiers"[DE BLASIS, 1982]. On voit donc immédiatement l'importance de l'aspect visuel dans le processus de recherche .

Dès lors, l'hypothèse de travail est : qu'en permettant à un utilisateur de visualiser des en-têtes (sur un écran d'ordinateur) comme si le tiroir contenant ses dossiers était ouvert devant lui, il pourra faire l'association entre ce qu'il recherche et la représentation de celui-ci à l'écran. Cette approche présente l'avantage d'offrir une stratégie voisine de celle que l'on utilise soi-même pour rechercher quelque chose dans ses dossiers.

Ce qu'il faut retenir de cette section, en plus de l'approche du terme "bureautique", c'est, d'une part, l'importance du domaine sur lequel s'étend la bureautique et, d'autre part, la limitation de notre étude aux problèmes de stockage et de rangement.

Dans la section suivante, nous proposons de décrire brièvement deux applications en bureautique dans lesquelles des problèmes de stockage et de modélisation de l'information ont été rencontrés.

I.2 . LES APPLICATIONS ET LES BASES DE DONNEES EN BUREAUTIQUE

Cette section se propose de faire un bref état des lieux de l'intégration des SGBD dans le domaine de la bureautique.

I.2.A. LE DOCUMENT GÉNÉRALISÉ

L'emploi d'une base de données en bureautique a pour objectif le stockage, l'accès et la manipulation de documents. Le terme document est pris ici au sens large ; il peut contenir à la fois du texte, des graphiques, des images fixes ou animées, du son,.... Certains auteurs emploient dès lors le terme de document électronique ou **document généralisé.** Le terme d'information multi-média est également utilisé, et plus précisément dans un environnement hypertexte.

"Hypertexte est par essence un texte non linéaire, non séquentiel. Dans un système hypertexte, un document se présente sous la forme de **noeuds** reliés entre eux par des **liens** (pointeurs). Un noeud se définit comme un texte mais aussi comme des images, du texte et des graphiques combinés, du son digitalisé ou encore des images animées. Un lien peut pointer sur une position précise d'un noeud, sur un espace du noeud ou bien encore sur l'entièreté du noeud. Ce sont ces liens entre les noeuds qui donnent au document sa caractéristique de non-linéarité. En effet, ils rattachent des noeuds appartenant à un même document ou à des documents différents. L'ensemble des documents d'un système hypertexte peut être vu à son tour comme un hyperdocument que l'on peut parcourir grâce aux liens.[DELISLE, SCHWARTZ, 1986]

Actuellement il existe quelques systèmes hypertexte.(Hypercard sur MacIntosh par exemple) Pour une étude approfondie, nous renvoyons le lecteur intéressé à [DELISLE, SCHWARTZ, 1986] et à [CONKLIN, 1987].

Un objet tel qu'un document généralisé n'est pas directement gérable par les SGBD "conventionnels" car plusieurs problèmes se posent. Notre propos est, ici, de répertorier quelques uns de ces problèmes [ADIBA, 1986] dont certains apparaîtront également dans notre travail.

Les problèmes qui se posent sont multiples . Ils concernent entre autres:

- la description de ces documents généralisés qui sont de nature complexe. Chaque type de document possède une structure logique qui lui est propre. Par exemple, un mémoire est composé d'une page de garde, d'un plan, de chapitres qui sont eux-mêmes stucturés en sections qui elles-mêmes sont....et ainsi de suite. Le système doit connaître cette structure afin d'apporter une aide à l'utilisateur dans la manipulation de son document. De plus, lors de la création du document , il faut pouvoir passer aisément de l'éditeur de texte (lorsque le mode de représentation est le mode écrit) à la base de données en respectant la structure désirée pour ce document;

- la manipulation de la base de données ce qui entraîne des problèmes de langage. Il faut ainsi définir des langages adaptés au contexte de la bureautique, des fonctionnalités propres à ce contexte et des interfaces sophistiquées (importance de l'aspect visuel);

- le stockage et l'accès aux documents généralisés. Outre le fait que ces objets peuvent avoir une structure complexe, leur taille est généralement importante surtout s'ils sont multi-média. Ces tailles importantes (une image demande beaucoup de ressource mémoire) conduisent à des problèmes de stockage et impliquent un matériel de mémorisation de grande capacité tel que, par exemple, le disque optique numérique;

- le partage d'un même document entre plusieurs utilisateurs (notion d'emprunt,...).

- enfin, un problème, tout aussi important, est celui qu'implique l'utilisation de telles techniques au niveau organisationnel et humain.

Ce bref aperçu montre que les problèmes soulevés ne sont aisés à résoudre. Ainsi, les solutions envisagées pour l'intégration de documents généralisés dans les SGBD peuvent s'inscrire dans trois approches :

- la première consiste à étendre les fonctionnalités internes d' un SGBG existant . Cette approche a l'avantage de garder toutes les fonctionnalités du SGBD et permet une adaptation aux comportements spécifiques des données manipulées;

- la deuxième envisage la construction d'un nouveau SGBD capable de manipuler des documents généralisés. Il s'agit de la solution la plus intégrée mais aussi la plus coûteuse;

- enfin la troisième approche prône la construction d'une couche spécialisée au-dessus d'un SGBD existant . Le principal avantage est d'obtenir ainsi une indépendance vis-à-vis du SGBD de base.

Bien que, rappellons-le, nous ne manipulerons pas le contenu des documents (un document sera considéré comme un tout indivisible), c'est cette dernière approche que nous emprunterons . Notre but sera donc de construire une couche (ici un ensemble de primitives) au-dessus d'un SGBD en faisant en sorte que celui-ci soit invisible à l'utilisateur de nos primitives.

Notre propos est donc similiaire à celui du Projet BIG de Toulouse et du Projet TIGRE de Grenoble, deux projets importants de par le nombre de personnes et de matériel mobilisés. De ces projets que nous allons décrire, nous espérons retenir des similitudes, des idées voire des solutions qui pourront nous aider dans la réalisation de notre mémoire .

Vu l'étendue de ces projets, nous limiterons notre intérêt à l'aspect descriptif des objets complexes manipulés ainsi qu'aux aux solutions proposées.

I.2.B. LES APPLICATIONS EN BUREAUTIQUE : TIGRE ET BIG

1). le projet TIGRE

L'objectif du Projet TIGRE [VELEZ,1986] est de fournir un ensemble d'outils permettrant de manipuler des documents (ou des parties de document), de les retrouver et de les insérer dans une base de données.
L'approche suivie dans ce projet conduit à la construction d'une couche spécialisée au-dessus d'un SGBD existant.

La manipulation d'un document généralisé a nécessité la définition d'un modèle de données qui tient compte de la structure logique et des composants de nature diverse des documents multi-média. Le modèle de base utilisé est le modèle Entité/Association auquel des extensions telles que l'**agrégation**, la **généralisation** et des **abstractions** ont été apportées. Le modèle résultant est appelé TIGRE.

Dans ce modèle, un document consiste en un **objet typé**, existant en tant que valeur d'une entité ou d'une association. Le type document est construit par le constructeur (*) Document. Quant à la structure hiérarchique, elle est bâtie à l'aide d'opérateurs de composition qui sont vus comme des sous-constructeurs du constructeur Document Ces sous-constructeurs permettent de définir des éléments de structure à partir d'objets. Ces objets peuvent être soit des composants élémentaires (appelés unités et de nature diverse : texte, graphique, ..., **référence** à un autre élément,...), soit des éléments de structure , soit des structures de type document.

(*) Un constructeur est une des opérations qui permet de construire n'importe quelle occcurrence de type par compositions successives de ses constructeurs . Il définit également un ensemble d'opérations qui peuvent s'appliquer aux objets des types générés par lui.
Exemples simples de constructeurs : tableau, enregistrement, ... en PASCAL.

Afin d'illustrer la technique utilisée, voici un exemple extrait de [VELEZ,1984]. Ici, les sous-constructeurs employés permettent une mise en séquence d'objets (Begin...End), un choix entre plusieurs objets (case...of..), et la création d'une liste d'objets d'une même classe (List (min,max) of...).

```
type Proceedings : document
      structure
            Begin
              .
              .
            Sommaire : List (1,*) of texte;
            corps : List (1,30) of. cas type-article of
                        article-complet : rapport-recherche.structure;
                        résumé : begin
                                    auteur : texte;
                                    titre : texte;
                                    contenu : List (1,5) of
                                          paragraphe.structure;
                                 end;
                        end;
              .
              .
            end;
```

Le SGBD TIGRE repose sur un SGBD relationnel ce qui implique la réalisation d'une correspondance entre le modèle TIGRE et le modèle relationnel. Physiquement, les règles de définition de type ainsi que les occurrences de document sont stockées sur plusieurs relations. On remarquera que les objets volumineux sont découpés en fragments dont la taille correspond à la taille maximale de n-uplets du SGBD sous-jacent et stockés comme des n-uplets. On notera également que, selon les capacités de stockage de ce SGBD sous-jacent, on décide de stocker des objets de grandes tailles soit dans la base de données, soit dans des fichiers externes.

Dans le SGBD TIGRE chaque document est identifié par un **identificateur interne** délivré par le système.

Enfin, au modèle TIGRE on a associé un langage de manipulation (langage LAMBDA) qui propose des énoncés pour parcourir la structure interne des documents, et des opérations spécifiques au type document. De par le fait qu'elle se base sur la notion de type , l'intégration de LAMBDA dans un langage de programmation constitue une tâche relativement simple.

Comme nous le verrons plus loin dans ce travail, nous avons retenu certaines techniques et approches du modèle TIGRE. Ainsi, nous utiliserons un identificateur interne et le choix du stockage en base de données sera laissé à l'utilisateur.

I.2) Le projet BIG

La deuxième application à laquelle nous porterons un intérêt est celle réalisée dans le cadre du Projet BIG [CHRISMENT et al, 1986] . Ici aussi, l'utilisation du modèle relationnel comme modèle sous-jacent pose le problème de la limitation dans la capacité de description des données. Le modèle relationnel ne permet, en effet, "que" de décrire des informations faiblement structurées et constituées d'éléments dont les valeurs sont peu volumineuses. Ces limites du pouvoir de description proviennent, d'après les auteurs, de la normalisation (mise sous formes normales d'une relation), qui peut provoquer la décomposition d'information formant un tout au point de vue sémantique, et des opérateurs relationnels qui exigent des opérandes de faible taille .

Un document généralisé étant un objet volumineux et fortement structuré, de définir de nouveaux modèles de description de données se révéle nécessaire, ainsi que nous l'avons vu précédemment.

Le modèle construit pour répondre à ces nouvelles contraintes de structuration et de taille en matière de base de données est le **modèle Agrégatif.** Ce modèle permet la généralisation de la notion d'information grâce au concept de **type abstrait** (*) de données, concept qui permet de définir des informations des plus élémentaires aux plus complexes.

Ce modèle Agrégatif repose sur les types suivants :

- les types de base, types qui ne sont définis à partir d'aucun autre. Ces types sont les types communément employés dans les SGBD (entier, réel, caractères,...);

- les types structurés décrivant des documents. Ces types sont constitués d'un ensemble d'éléments (attributs) appartenant à des types éventuellement distincts.

(*) "Un type abstrait correspond à un concept du problème que l'on résout et est caractérisé par des propriétés structurelles et par des spécifications des opérations associées" [VAN LANSWEERDE, 1988].

Exemple extrait de [CHRISMENT,1986] :

```
Type TVA
    record
        CODETVA  : INTEGER *; (* désigne un identifiant)
        TAUXTVA  : STRING;
    end record;

PRODUIT
  record
     NOP  : INTEGER *;
     DESIGNATION  : STRING;
     TAXE  : TVA;
  end record;
{ on remarque que le type PRODUIT est défini notamment par un
  attribut lui-même structuré };
```

- les types de collections où l'on distingue :
  - les ensembles
    (exemple : type Ensemble caractères : set ofCHAR) ;
  - les relations : ce sont les relations telles que définis dans le modèle relationnel.(exemple : Type CATALOGUE : relation of produit);
  - et les tableaux.

Enfin, les associations entre les informations sont spécifiées par la définition de contraintes d'intégrité exprimées dans une syntaxe proche de SQL.

L'utilisation du modèle relationnel engendre un autre problème. En effet, tout type structuré doit respecter les formes normales décrites dans le modèle relationnel de CODD.

Un document se distingue cependant des autres types structurés dans la mesure où l'ordre des attributs est significatif (respect de l'ordre logique des composants d'un document). Cette contrainte empêche la décomposition du type structuré décrivant le document. Dès lors, il existe dans la base de données des valeurs redondantes qu'il faut gérer en pausant des contraintes d'intégrité.

Signalons enfin, qu'une autre gestion a été rendue nécessaire lorsque les valeurs d'attributs sont volumineuses.

Conclusions

Dans ce premier chapitre, nous avons défini le contexte dans lequel nous travaillons (le contexte de la bureautique) et les concepts qui lui sont associés tels que le bureau, l'information et le document généralisé. Nous avons également précisé les limites de notre travail à savoir : les fonctions de classement et de stockage.

Deux applications en bureautique ont attiré notre atttention. De leur examen, nous retenons la nécessité de définir de nouveaux modèles de description de données, les SGBD conventionnels montrant une certaine "lacune" dans la modélisation d'objets volumineux et complexes. Parmi les approches suivies pour combler cette "lacune" et pour adapter l'emploi de bases de données au contexte de la bureautique, certaines vont nous aider à résoudre notre problème. Nous avons retenu ainsi des techniques et solutions telles que :

- la construction d'une couche spécialisée au-dessus d'un SGBD existant;
- l'emploi de type abstrait pour modéliser des objets complexes et structurés;
- l'utilisation d'un identifiant interne permettant de distinguer les objets au sein du système;
- la liberté du choix de stocker l'information dans la base de données, liberté laissée à l'utilisateur averti da la taille de l'objet.

Enfin, il est apparu que le modèle relationnel ne semble pas tout à fait adéquat pour modéliser des structures complexes puiqu'il faut entre autres assurer une gestion supplémentaire via des contraintes d'intégrité (Projet BIG) et que les valeurs volumineuses engendre également un problème de gestion.

Si nous nous sommes plus particulièrement attaché à l'aspect de description des données c'est parce que nous manipulons des objets de bureau, objets complexes et pouvant être volumineux. Ces objets font l'objet d'une étude présentée dans le chapitre suivant.

Chapitre II : ANALYSE FONCTIONNELLE

INTRODUCTION : L'APPROCHE ORIENTÉE OBJET

La démarche que nous adoptons est la suivante : l'étude d'un ER manuel nous permettra d'isoler les objets de bureau et les opérations qui leur sont attachées. Ce sera l'objet de la section 1.

Ensuite, nous réaliserons le passage vers un environnement de rangement automatisé en ayant soin de conserver le maximum de fonctionnalités du bureau manuel mais aussi avec le souci d'apporter toute une série d'améliorations. Ce sera l'objet de la section 2.

La description faite au cours de l'analyse fonctionnelle est réalisée au moyen de l'approche orientée objet. C'est pour cette raison qu'il nous parait intéressant de présenter les caractéristiques fondamentales de cette dernière [MEYER, 1988]
Dans un premier temps, nous nous attachons à définir les concepts de base de l'approche pour en exposer ensuite les grands principes.

A. LES CONCEPTS DE BASE

Nous allons passer en revue les différents concepts sous-jacents à l'approche et nous les illustrerons par des exemples.

Le premier concept est celui **d'objet**. Le monde réel se compose d'éléments divers appelés objets. Ceux-ci sont entièrement définis par les propriétés qui leur sont inhérentes et par les opérations qui leur sont associées. Les caractéristiques d'un objet reprennent à la fois ses propriétés et ses opérations. Nous nous retrouvons ici en présence d'une application de la notion de **type abstrait des données** [VAN LANSWEERDE, 1989].
Cette notion d'objet est, cela va sans dire, au centre de l'approche orientée objet.

Le deuxième concept est celui de **classe**. Il s'agit d'une description générale qui est en mesure de caractériser un ensemble d'objets semblables. Un objet appartenant à une classe est une instance de celle-ci. Toutes les instances d'une classe partagent les mêmes caractéristiques. Les représentants d'une classe se différencient suivant la valeur d'une variable, appelée variable d'instance. Par exemple, nous pouvons définir la classe Polygone avec les propriétés position, couleur et les opérations Calculersurface, Deplacervers(a,b) et Pointcentral.

Le terme classe est parfois remplacé par celui de **type** et l'on parle alors de types d'objet et de définition de type.

Le troisième concept est celui de **message**. C'est grâce à lui que l'on va pouvoir exécuter des opérations sur un objet. Le message sera envoyé par un objet émetteur à destination d'un objet récepteur. Ce dernier déterminera lui-même, en fonction du message reçu, les actions qui doivent être choisies afin d'exécuter l'opération. L'ensemble des actions choisies est appelé **méthode**.

Cette dernière notion terminait la section consacrée à la définition des concepts de l'approche orientée objet, maintenant nous allons présenter les principes généraux de la méthode.

B. Les grands principes de l'approche orientée objet

Le premier principe veut que la modularisation dans une approche orientée objet se fasse sur base des structures de données et non pas sur base des fonctionnalités. En effet, au sein d'un système, les fonctions se modifient sans cesse tandis que les propriétés des objets concernés par les opérations n'évoluent quasiment pas.
Dès lors, il vaut mieux choisir une décomposition du système selon les classes d'objets sur lesquelles ce dernier va travailler que suivant les fonctionnalités qu'il doit remplir.

Selon le deuxième principe, la spécification des opérations doit être totalement indépendante de la représentation interne de l'objet sur lequel elles travaillent. Ce principe d'**abstraction des données** est atteint en ne permettant la manipulation des propriétés d'un objet qu'à partir des opérations qui leur sont attachées.

Le troisième principe concerne les classes d'objet et préconise le regroupement de tous les objets similaires au sein d'une classe (cfr section A).

Le quatrième principe définit le mécanisme d'**héritage**. Combiné avec la notion de classe, il permet d'établir la relation "sous-classe de" entre deux classes. Cela signifie que si A est sous-classe de B, alors A, en plus de ses caractéristiques propres, hérite de toutes les propriétés et opérations de B; cette dernière sera appelée la **super-classe** et A la **sous-classe**. Notons que les caractéristiques héritées peuvent être redéfinies ou inhibées.
Cette relation est très importante car elle autorise la création d'une nouvelle classe en lui ajoutant les caractéristiques d'une classe déjà existante.
Si l'on prend la classe Polygone qui donne la description des polygones, on pourrait établir une relation "sous-classe de" entre la sous-classe Carré et la super-classe Polygone.

L'approche permet à une classe de partager plusieurs relations "sous-classe de" avec d'autres classes; nous sommes alors en présence du principe de l'**héritage multiple.**

Le dernier principe concerne une propriété de la variable d'instance : le **polymorphisme.** En effet, cette dernière peut désigner (référencer) des instances de classes différentes.

Cette approche ne sera suivie que lors de l'analyse fonctionnelle, nous l'abandonnerons lors de la phase d'implémentation.

CONCLUSION : LES INTERETS D'UNE APPROCHE ORIENTEE OBJET

L'approche orientée objet permet la distinction entre les données (propriétés des objets) et les opérations (comportement des objets). Pour ce faire, elle intègre des mécanismes de généralisation (définition de classes) et de spécialisation (relation entre super et sous-classe), ce qui donne à l'approche une puissance de modélisation accrue.

La modélisation qu'elle autorise de la réalité est plus naturelle et donc plus accessible. Il n'y a pas vraiment de fossé sémantique entre les objets et leur représentation.

En outre, elle offre une indépendance entre la sémantique des objets (c'est-à-dire leurs propriétés et leur comportement) et leur représentation. Dès lors, les changements de représentation d'un objet ou de l'implémentation des opérations n'ont pas d'impact sur les programmes qui utilisent cet objet.

Enfin, l'approche orientée objet permet d'augmenter la productivité du programmeur et d'écourter le cycle de vie d'un projet. En effet, de nombreux types génériques peuvent être prédéfinis et réutilisés par après.

II.1. DESCRIPTION D'UN ENVIRONNEMENT DE RANGEMENT D'INFORMATIONS

Tout d'abord, nous décrirons successivement les informations (ou documents) et les meubles de rangement d'un ER; ensuite, nous donnerons un aperçu de la structure d'un ER et enfin, nous envisagerons les propriétés et les opérations des objets, ce qui nous permettra de les regrouper en classes.

II.1.A. DESCRIPTION DES OBJETS D'UN ENVIRONNEMENT DE RANGEMENT MANUEL

L'établissement de la liste des objets de l'ER se fera de bas en haut c'est-à-dire du contenu vers le contenant, de l'objet le plus particulier (le document) au plus général (le bureau). Pour ce faire, nous sommes amenés à distinguer les objets suivant [DACHOUFFE,1986] :

- le **document** qui est une information spécifique représentée sous forme écrite, graphique ou mixte. Chaque mode de représentation a ses propres types : le mode écrit se compose, par exemple, des types suivant : formulaire, message, mémo, fiche, ..., mais quel que soit son mode de représentation, le document est caractérisé par un en-tête et un contenu (cfr Chapitre I : La bureautique);
- la **fiche** qui est un document particulier sur lequel on inscrit des renseignements et qui renvoie celui qui le lit à un autre objet;

- la **chemise** qui est une enveloppe cartonnée ou toilée, servant à classer des documents concernant une personne, une affaire.
Le contenu de l'enveloppe est désigné sous le nom de **dossier**. Dorénavant, nous utiliserons indifféremment les termes "chemise" et "dossier";

- la **boîte** qui est un coffret de carton ou de matière plastique, facilement transportable et généralement muni d'un couvercle. Elle peut renfermer des chemises et des documents;
- la **pile** est quant à elle une boîte découverte, un bac où sont amassés des documents ou des chemises placés les uns sur les autres;

- le **répertoire** qui est une boîte contenant un inventaire méthodique où les matières sont classées dans un ordre qui permet de les retrouver facilement (par exemple, un répertoire d'adresses).
Si celui-ci est un ensemble de fiches classées dans un certain ordre et permettant de retrouver aisément des documents ou des dossiers, alors il prend le nom d'**index**. Chacune des fiches reprend au minimum les coordonnées du dossier ou du document dont elle est une référence;

- la **poubelle** qui est un récipient destiné à recevoir des objets du bureau dont on veut se débarrasser; tant qu'elle n'a pas été vidée, les documents ou les chemises de la poubelle peuvent être récupérés sans trop de difficultés;

- le **plan de travail** qui est un meuble nécessaire à la création d'un document ou à la manipulation d'autres objets comme une chemise ou une boîte;

- l'**étagère** qui est un meuble, le plus souvent ouvert, formé de montants qui soutiennent des **tablettes** horizontales disposées par paliers; ses compartiments peuvent abriter des documents, des chemises ou des boîtes;

- l'**armoire** qui est une meuble haut et fermé par des battants, composé de tiroirs il est destiné à stocker des objets divers;
- les **tiroirs** qui sont des compartiments coulissant s'emboîtant dans un emplacement réservé; ils peuvent recevoir des documents, des dossiers ou des boîtes.

- enfin, le **bureau** que nous envisageons comme un lieu de travail (cfr Chapitre I : La Bureautique) regroupant des meubles de rangement et des documents.

Tous les objets qui ont été isolés dans la description de l'ER correspondent à autant de **classes** ou de **types.** C'est ainsi qu'apparaissent les classes Document, Répertoire, Chemise ou Dossier, Boîte, Pile, Armoire et ainsi de suite jusqu'à la classe Bureau.

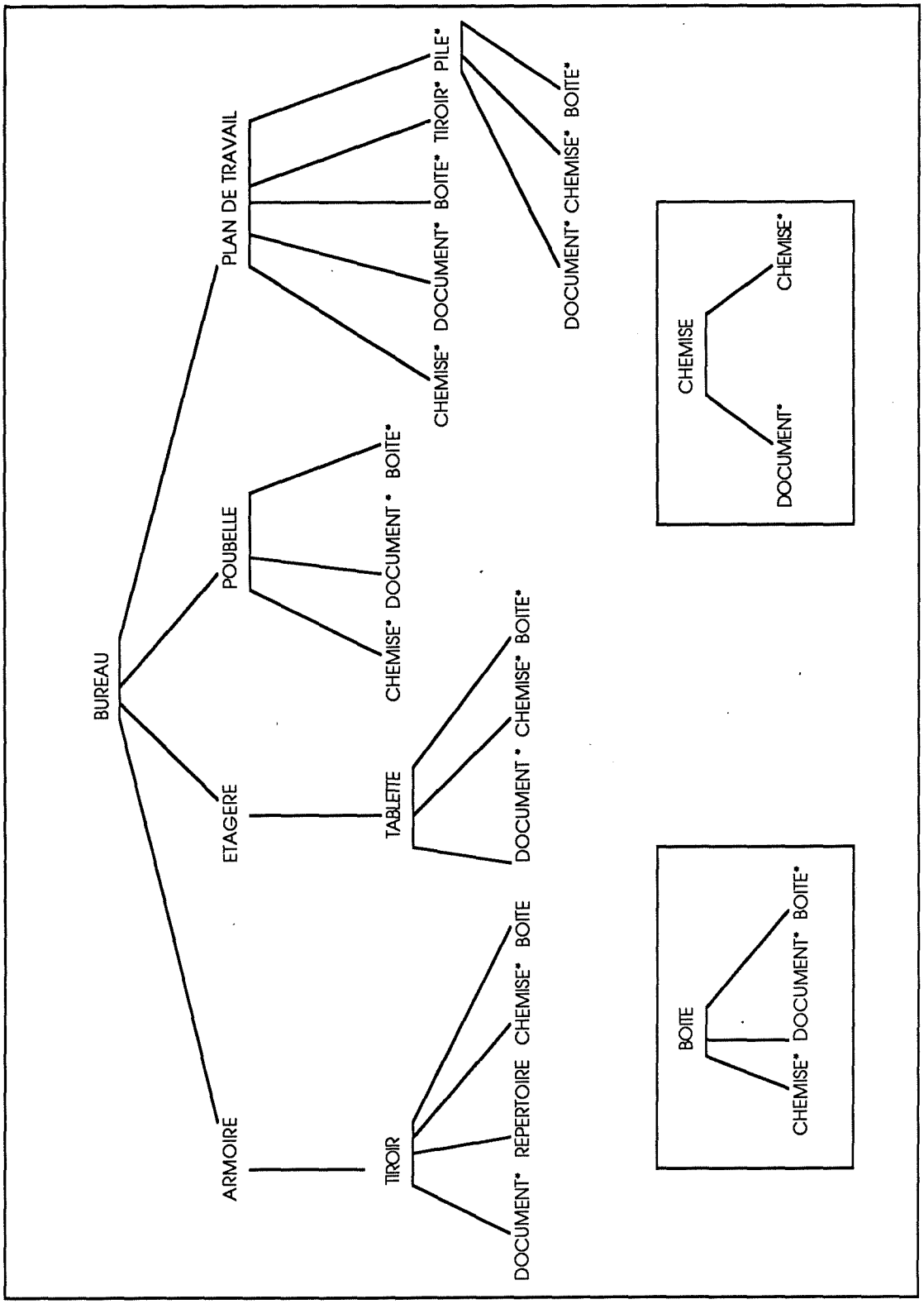
Avant de définir leurs caractéristiques, il nous a paru nécessaire de préciser les relations d'inclusion qui peuvent exister entre les différentes classes d'objet dont nous venons d'établir la liste.

II.1.B. STRUCTURE D'UN ENVIRONNEMENT DE RANGEMENT

Il y a deux raisons à la présence de cette sous-section. D'une part, il est temps de donner une image de la structure d'un environnement de rangement manuel et d'autre part, grâce à cette section, le lecteur pourra déjà prendre conscience de l'importance des **relations d'inclusion** entre les types d'objet au sein d'un ER.

Qui dit structure, dit éléments mais aussi relations; dans le cas qui nous préoccupe, les éléments seront représentés par les objets de l'ER et les relations par les relations d'inclusion entre les représentants des classes.
Nous parlerons ici de relation directe et il est bien entendu qu'un document peut se retrouver dans une armoire mais uniquement de façon indirecte et cela par l'intermédiaire d'un tiroir.

C'est le schéma I.1 qui va nous permettre de représenter la structure générale d'un ER; il reprend tous les objets d'un ER et se présente sous la forme suivante :



II.1.C. PROPRIÉTÉS DES CLASSES D'OBJET

Nous n'avons pas pour objectif d'énoncer une liste exhaustive des propriétés de chaque classe d'objet, mais nous estimons que la liste proposée est suffisante pour caractériser précisément les objets de l'ER manuel.

1) Définitions

Les définitions portent sur des propriétés qui, sans précision, risquent d'être vagues voire ambiguës pour le lecteur.

Remarquons d'abord que par **description** d'un objet, nous entendons un ensemble de renseignements divers tels que ses caractéristiques physiques (taille, volume, couleur, matériau, ...), les personnes qui le manipulent, les autorisations d'accès, son équipement, ...

Le **mode de classement** est la façon dont les objets sont rangés au sein de l'objet qui les contient. Nous distinguons le classement croissant et décroissant, sur des nombres ou des caractères.

D'autre part, nous opérons également la distinction entre le **contenu** d'un objet, c'est-à-dire les documents et/ou dossiers et/ou fiches contenus directement ou indirectement dans l'objet et qui apportent à celui-ci une valeur sémantique (par exemple, l'armoire qui contient les dossiers médicaux) et les **types d'objets qu'il peut recevoir** (pour en avoir une liste exacte, on se reportera aux relations d'inclusion directe).

2) Les propriétés des classes

Si l'on se rapporte à la description faite dans l'ER manuel, nous remarquons d'emblée la similitude qui existe entre les classes d'objet.
C'est pour cette raison que nous définissons une classe d'objet tout à fait générale, celle des **OBJETS DE BUREAU** qui possède toutes les propriétés générales des types d'objets de bureau isolés précédemment.

Les propriétés de cette classe sont :
- le **nom** de l'objet de bureau;
- le **créateur** ou le **responsable** de l'objet de bureau;
- la **date de création** de l'objet;
- la **date de dernière mise à jour** de l'objet;
- la **description** de l'objet de bureau.

A partir de cette classe générale, nous sommes en mesure d'isoler deux sous-classes : la sous-classe des **OBJETS DE RANGEMENT** et la sous-classe des **OBJETS RANGEABLES**. Celles-ci rassemblent respectivement les caractéristiques des objets de rangement et des objets rangeables.

En plus des propriétés héritées de leur super-classe (OBJETS DE BUREAU), les deux sous-classes citées possèdent des propriétés spécifiques.

Les propriétés spécifiques à la sous-classe OBJETS DE RANGEMENT sont :

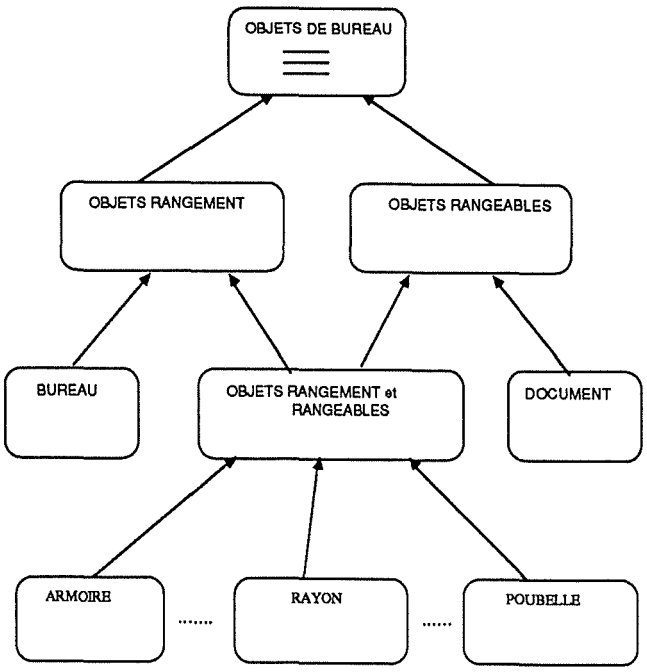
les types d'objet qu'il peut recevoir;
son **mode de classement**;
la caractéristique sur laquelle se fait le classement de ses composants **(propriété classement)**

La sous-classe OBJETS RANGEABLES se caractérise par son lieu de rangement (sa **localisation**) et deux **propriétés additionnelles** dont le but est de permettre un classement autre que sur les propriétés déjà définies;

Il faut noter que la plupart des classes isolées au point A sont des sous-classes de OBJETS RANGEABLES et OBJETS DE RANGEMENT : c'est le cas des classes Armoire, Etagère, Tiroir, Tablette, Chemise, Boîte, Pile, Poubelle et Plan de travail.
On peut alors créer la nouvelle classe des **"OBJETS DE RANGEMENT ET RANGEABLES"** : sous-classe des classes OBJETS RANGEABLES et OBJETS DE RANGEMENT

Enfin, la classe Document possède des propriétés qui lui sont propres; il s'agit de son **mode de représentation**, de son **type**, de son **contenu** ainsi que de sa **référence à l'original**.

Voici une représentation des différentes classes d'objet que nous venons de définir :



II.1.D. LES OPÉRATIONS SUR LES CLASSES D'OBJET

L'énumération des opérations se fait pour chaque classe d'objet et les opérations spécifiques à une classe ou sous-classe sont (re)définies à part.

Les opérations que l'on peut associer à la classe générale des OBJETS DE BUREAU sont :

la création de l'objet;
la localisation et l'accès à l'objet;
l'ouverture de l'objet;
la modification d'une des propriétés de l'objet;
la destruction de l'objet;
la fermeture de l'objet.

Les opérations spécifiques à la classe des OBJETS DE RANGEMENT sont les suivantes :

l'ajout ou le retrait d'un composant à l'objet;
prendre connaissance des composants de l'objet;
vider l'objet de son contenu.

Les opérations associées à la classe des OBJETS RANGEABLES sont :

le rangement de l'objet tout en respectant le mode de classement en vigueur;
le transfert d'un objet de bureau d'un ER à un autre.

Il est bien entendu que la classe des OBJETS RANGEMENT ET RANGEABLES hérite de toutes les opérations associées aux classes OBJETS RANGEABLES et OBJETS DE RANGEMENT.

L'opération "retirer ou ajouter un composant à l'objet" pour la classe Pile est redéfinie; elle ne peut se faire que sur l'élément qui se situe en son sommet.

De même, la classe Document se voit associer une opération complémentaire : celle de la copie. Notons en passant qu'il y a aussi pour les documents toutes les opérations relatives au contenu (consultation, modification, ...); nous ne nous y attardons pas puisque notre but n'est pas de travailler sur le contenu des documents mais bien de les ranger.

CONCLUSION

Que faut-il retenir de cette description ?

Premièrement, la définition de la classe des **OBJETS RANGEMENTS ET RANGEABLES** qui regroupe la plupart des objets de l'environnement de rangement et ensuite l'importance des **relations d'inclusion** entres les différents types d'objet de l'ER.

II.2 LE PASSAGE À UN ENVIRONNEMENT AUTOMATISE

En réalisant ce passage, nous voulons conserver le maximun des fonctionnalités décrites dans l'environnement de rangement manuel. Nous espérons en ajouter d'autres grâce à l'aspect d'automatisation. Signalons que ces fonctionnalités se concrétisent sous la forme d'opérations offertes à des programmeurs d'application en bureautique. Dès à présent, nous tiennons compte du fait que l'automatisation de la fonction de rangement repose sur l'utilisation d'une base de données.

La présente section se décompose comme suit : dans un premier temps nous précisons les objets que nous manipulerons et nous émettons quelques considérations à leur sujet. Ensuite, nous envisageons les problèmes engendrés par l'automatisation et l'utilisation d'une base de données. Enfin, nous donneons les apports découlant de l'automatisation, en insistant particulièrement sur la possibilité de structuration d'un environnement de rangement automatisé.

II.2.A LES OBJETS MANIPULES

Parmi les types d'objets isolés dans l'étude précedente, nous avons en retenons certains, modifions quelques uns et regroupons d'autres, à savoir :

1). Le document

Le type d'objet fondamental repris est le type **Document**. En effet, c'est grâce aux instances de ce type que de l'information sera effectivement stockée. Ranger de l'information équivaut donc à ranger l'instance du type Document contenant cette information.

par le passé (cfr Chapitre I : La bureautique), nous considérons un document comme la réunion d'un en-tête et d'un corps. Cette subdivision se justifie d'autant plus que, dans l'automatisation, seul l'en-tête se révéle indispensable pour le rangement et l'accès au document.

Pour le problème qui nous préoccupe, nous distinguons deux types de document :

- les documents dits électroniques (cfr Chapitre I : La bureautique) c'est-à-dire ceux qui existent sous forme informatique et dont le corps peut être stocké dans une base de données ou sur un support de masse quelconque sous la forme d'un fichier;

- les documents non électroniques c'est-à-dire ayant une existence physique autre qu'informatique (un livre, une bande sonore...).

Cette distinction nous impose l'ajout d'une nouvelle propriété au type document : la propriété ELECTRONIQUE.Elle permet de préciser si le document est électronique ou non.

2) Le bureau

Ce type d'objet est retenu du fait qu'il contient - directement ou indirectement - tous les autres objets.

3) le plan de travail

Comme nous l'avons vu précédemment dans un ER manuel, le plan de travail constitue le passage obligé pour que certains objets tels que le document, la chemise, la boîte,... puissent devenir **COURANT.** Cet état courant se révéle nécessaire afin de pouvoir travailler sur l'objet.

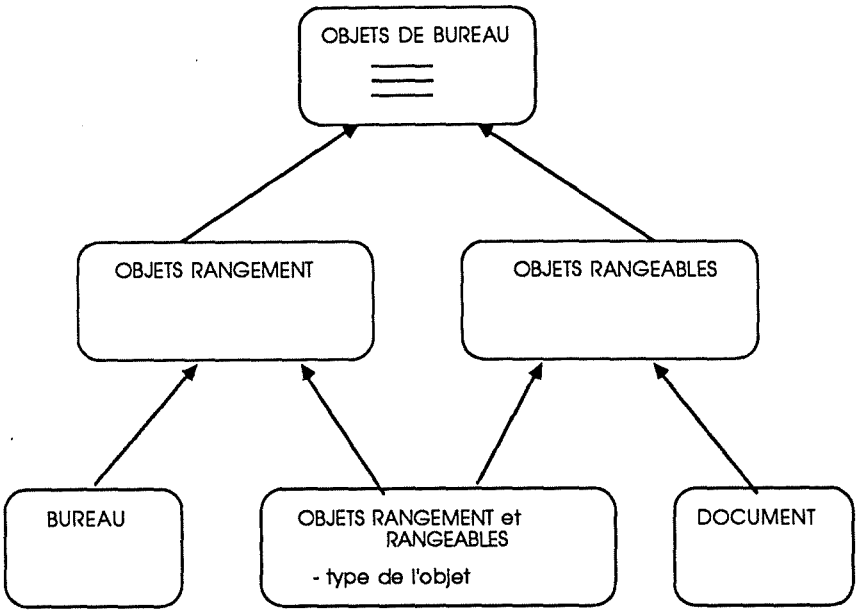
Dans un ER automatisé, ce passage n'est plus obligatoire pour obtenir l'état courant (voir C. Les apports de l'automatisation) et le plan de travail perd ainsi sa fonctionnalité propre. Il conserve cependant une utilité : celle d'un objet de rangement sur lequel sont présents des documents, des dossiers sur lesquels on travaille fréquemment et qui jouissent ainsi d'un accès aisé.

4) Les objets rangeables et de rangement

Lors de l'étude d'un ER manuel, nous avons créé quatre classes d'objet : une classe générale, celle des objets de bureau (OBJETS DE BUREAU); deux sous-classes, celle des objets de rangement (OBJET DE RANGEMENT) et celle des objets rangeables (OBJETS RANGEABLES); enfin, une classe des objets à la fois rangeables et de rangement (OBJETS DE RANGEMENT et RANGEABLES). A l'exception du Bureau et du Document , toutes les classes isolées dans l'ER manuel appartiennent à la classe des objets de rangement et rangeables.

Dans un ER automatisé, il n'y a plus lieu de distinguer les objets des classes Pile, Boîte,..., Chemise, Armoire. En effet, la distinction des différents objets dans un cadre manuel, reposait uniquement sur leurs **caractéristiques physiques.** Or, celles-ci n'ont plus cours dans un cadre automatisé.

Il en résulte la disparition de toutes ces classes au profit d'une seule, celle des objets rangement et rangeables. La hiérarchie devient alors :



SCHEMA 2- .

Pour pouvoir encore distinguer une armoire, d'un tiroir, d'une étagère,.. nous introduisons une nouvelle propriété pour la nouvelle classe OBJETS RANGEMENT et RANGEABLES : la propriété TYPE_OBJET.

Tous les types d'objets à savoir : Bureau , Plan de travail, Armoire , Tiroir, Etagère, Rayon, Chemise, Pile, Poubelle,.. sont donc repris.

5) Quelques considérations sur les objets

- Nous pouvons considérer un objet de rangement comme une **information structurée** et donc le voir comme composé d'un en-tête et d'un corps. Exemple : une armoire a un en-tête dans lequel sont reprises toutes ses propriétés et un corps représentant un ensemble (éventuellement vide) de tiroirs.

- Les caractéristiques physiques des objets de bureau n'ont pas été retenues, leur apport étant, dans le cadre de l'automatisation, jugé négligeable. Ceci implique, entre autres, que le concept de volume ne sera pas repris. Il en découle, en conséquence, que tout objet de bureau sera extensible à volonté (dans l'acception "capacité à contenir").

II.2.B. LES PROBLEMES ENGENDRÉS PAR L'AUTOMATISATION

Ces problèmes, qui proviennent principalement de l'emploi d'une base de données, sont :

1) le stockage du document électronique

En effet, la première question qui se pose dans un environnement de rangement automatisé est de savoir s'il faut stocker le corps du document dans la base de données ou ne stocker que sa localisation, c'est-à-dire, les indices qui permettront de le retrouver.
La localisation serait, par exemple, le chemin d'accès au fichier (le corps du document) stocké en dehors de la base de données.

Nous avons vu dans le Chapitre I que ce problème se pose dans le cas d'applications travaillant sur le contenu (exemple : le Projet TIGRE). Il est encore plus d'actualité dans notre cas car nous travaillons sur des micro-ordinateurs alors, que dans les cas évoqués dans le Chapitre I, les applications disposent de ressources mémoire nettement plus importantes (stations de travail).

Aussi, nous adopterons la même philosophie que celle suivie dans le Projet TIGRE. La décision de stocker le corps du document dans la base de données (dans l'ER) sera laissée au libre choix de l'utilisateur. Ce dernier doit cependant être averti que la place mémoire mise à sa disposition se trouve limitée soit à une disquette, soit au disque dur suivant le support de la base de données.

Dans le cadre de notre mémoire, nous nous limitons volontairement à une seule base de données. Il n'y est donc pas possible de répartir des documents dans plusieurs endroits de stockage.

Le fait de laisser la décision de stockage à l'utilisateur entraîne l'ajout de deux nouvelles propriétés au type Document : la propriété STOCKAGE et la propriété NOM_FICHIER.

2) La gestion des documents électroniques et non électroniques

Le fait de ne pas stocker tous les documents en base de données nous amène à assurer une **gestion mixte**.

Par gestion mixte, nous entendons la gestion de documents électroniques, dont le corps est présent ou non dans la base de données et de documents qui ne peuvent être stockés électroniquement (exemple : le Moniteur Belge,un dictionnaire...).

Ce type de gestion pose un problème dont tout utilisateur devra être conscient. En effet, on s'expose à un **asynchronisme,** une discordance entre la situation reflétée par la base de données, et la situation réelle.

En effet, comme le corps de certains documents n'est pas présent dans la base de données, nous pouvons y accéder et le manipuler en dehors notamment de tout programme d'application utilisant les primitives que nous offrons. Dès lors, toute modification, déplacement,... de ces documents ne sera pas automatiquement renseigné dans la base de données (un document pourrait être détruit alors que la base de données le référence toujours).

Dans ces cas particuliers, à moins d'un retour au programme d'application pour répercuter les modifications, l'état reflété par la base de données diffère de l'état réel de l'environnement de rangement.

A l'inverse, si le corps du document existe dans la base de données, les primitives utilisées afin de manipuler celui-ci se chargent du maintien de la cohérence.

II.2.C. LES APPORTS DE L'AUTOMATISATION

Dans cette sous-section, nous présentons une nouvelle définition de l'état courant,.avant de passer en revue les opérations propres à l'automatisation .

1) la notion d'objet courant

Toute opération sur un objet de bureau ne peut se faire que si celui-ci est **courant**. Nous considérons, dans un environnement de rangement automatisé, qu'un objet sera courant dès qu'on y a accédé. Or, le résultat de l'accès à un objet de bureau correspond à l'identifiant interne de l'objet (cfr Chapitre I : le Profjet TIGRE). C'est pour cette raison que nous pouvons dire qu'un objet, dont on détient l'identifiant interne, est dans un état courant. Tout ceci constitue un raccourci par rapport aux opérations nécessaires dans un environnement de rangement manuel où il fallait accéder, déplacer l'objet sur le plan de travail et l'ouvrir pour le rendre courant.

L'état "courant" d'un objet se révéle donc indépendant de sa situation sur le plan de travail.

2) L'opération de copie d'un document

La copie d'un document constitue une opération qui était déjà présente dans le cadre manuel. Le passage à un ER automatisé a pour effet d'enrichir cette opération. Le document copié aura, bien entendu, une existence propre mais, en plus, il sera possible de connaître la date de la copie ainsi que le document à l'origine de cette copie (pour plus de renseignements, nous renvoyons le lecteur aux spécifications fonctionnelles). Deux propriétés du type Document sont donc ajoutées : DATE_COPIE et COPIE_DE.

3) le texte libre

Tous les objets de bureau, qu'ils soient de rangement, rangeables ou les deux à la fois, possèdent un ensemble de propriétés qui les caractérisent. Cet ensemble se trouve fixé une fois pour toutes ce qui implique une rigidité certaine. Aussi, il nous a semblé utile de permettre à cet utilisateur de commenter de manière tout à fait libre (càd sous la forme d'un texte non standardisé et dont la longueur n'est pas limitée) l'objet qu'il place dans l'environnement de rangement. L'ensemble de ces informations est contenu dans ce que nous appelons un **texte libre**.

Exemple de texte libre d'un dossier :

" ce dossier n'est pas complet, il manque le formulaire n°XX qui est attendu au plus tard pour le 30/03/89, ... ".

Une propriété supplémentaire vient donc s'ajouter à la classe des objets de bureau : la propriété TEXTE_LIBRE.

4) Place d'origine et place réelle

Dans tout environnement de rangement, un objet rangeable possède un lieu de rangement d'origine, c'est-à-dire, le lieu où on est en droit de s'attendre à le rencontrer. Ceci n'exclut pas que cet objet soit rangé autre part (mais en un seul lieu !) et que sa place d'origine reste la même .

Par exemple :

un dossier médical est rangé sur le plan de travail. Il s'agit là de sa place réelle mais ce dossier devrait normalement se trouver dans le tiroir réservé aux dossiers médicaux. C'est sa place d'origine. Lorsque ce dossier sera remis à sa place d'origine , les deux lieux de rangement, à savoir : le lieu d'origine et le lieu réel, seront confondus.

Ces considérations sur les deux lieux de rangement d'un objet rangeable - lieux éventuellement distincts (le lieu d'origine n'est plus alors que "théorique") - nous permettent d'offrir à l'utilisateur une opération qui consiste à remettre un objet rangeable à sa place d'origine (voir les spécifications fonctionnelles).

Ces considérations entraînent la modification de la propriété localisation de la classe OBJETS RANGEABLES . Elle est alors remplacée par "localisation d'origine" et "localisation réelle".

Ces mêmes considérations nous amène à envisager 3 états possibles pour chacun des objets rangeables contenus dans un objet de rangement :

- le premier de ces états est celui où l'objet est effectivement présent et à sa place d'origine;

- le second est celui où l'objet est présent mais l'objet de rangement visité n'est pas sa place d'origine;
- et enfin, le troisième état est celui où l'objet rangeable n'est pas présent mais où l'objet de rangement est bien sa place d'origine.

L'utilisateur pourra ainsi savoir si un objet courant est à sa place d'origine ou dans un autre endroit. Dans cette dernière éventualité, il pourra, via une opération (voir Sect. 3 les spécifications fonctionnelles) connaître sa place d'origine.

5) Les relations avec d'autre environnement de rangement

Jusqu'à présent, nous n'avons considéré qu'une seule occurrence d'environnement de rangement. Or, il est manifeste qu'un ER ne " vit" pas en vase-clos. Des échanges continuels s'effectuent entre ER et l'automatisation permet d'envisager une opération de transfert d'objet de bureau d'ER à ER.

La réalisation de cette opération implique, de notre part, la possibilité de pouvoir travailler sur deux bases de données, celles-ci étant ouvertes en même temps. Cette réalisation sera plus aisée si les objets transférés possèdent une même représentation dans les bases de données simulant les ER. Dans le cas contraire, un programme de conversion des structures de données échangées.se révélerait nécessaire.

Grâce à cette opération, on pourrait concevoir un transfert d'objet de bureau de n'importe quel type au moyen d'un support de transfert tel qu'une disquette .

6) La confidentialité

Le stockage du corps du document dans la base de données engendre un renforcement de la confidentialité puisque l'accès au document (au corps) ne peut se faire que via une primitive de restitution de fichier.

La confidentialité pourra être gérée de façon plus simple par l'ajout, en plus de sa propriété CONFIDENTIALITE, la propriété MOT DE PASSE.

7) Les opérations de structuration

Une autre facilité permise par l'automatisation concerne la structuration de l'environnement de rangement automatisé.

Dans l'étude de l'existant (cfr Ch II, sect 1), nous avons présenté une structuration générale. Il va de soi que cette structuration peut être modifiée, soit en retirant un ou plusieurs type d'objet (la pile n'est plus reprise par exemple), soit en limitant le nombre d'occurences d'un type d'objet, soit encore en ajoutant ou en supprimant des relations d'inclusions (exemple : le plan de travail ne contient plus de pile). Si la modification de la structure d'un ER manuel ne constitue pas toujours une opération aisée (déplacer une armoire,...), il n'en n'est pas de même dans un ER automatisé puisque les contraintes physiques ont disparu.

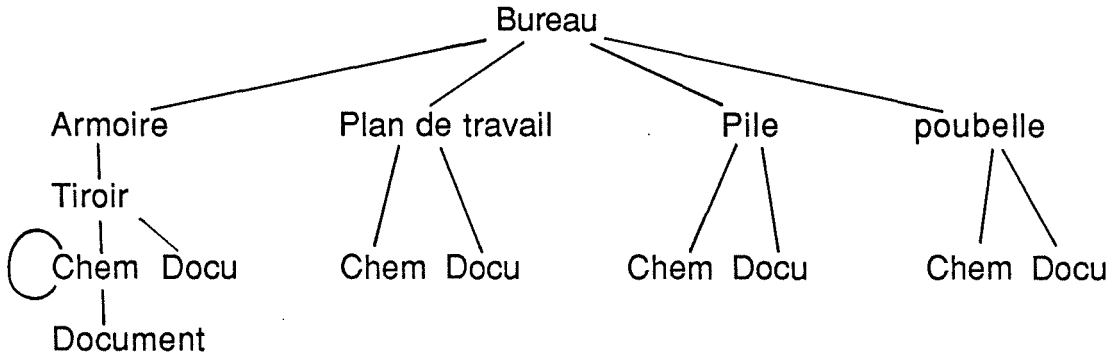
Pour exprimer la structuration d'un ER, on peut considérer les différents types d'objets (type Bureau, type Document,...) comme des **objets** ayant des propriétés et regroupés dans une classe appelée CATEGORIE OBJETS de BUREAU. Tous ces objets sont associés entre eux par des relations d'inclusion (inclusion au niveau logique). Les propriétés de ces objets sont :

- le type de l'objet :
  exemple : ARMOIRE (sous_entendu type Armoire);
- le nombre d'occurrences permises au sein de l'ER.;
- le nombre d'occurrences présentes dans ce même ER.

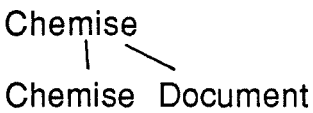
A cette nouvelle classe, nous pouvons associer les opérations suivantes :

- création d'un objet (c'est-à-dire création d'un nouveau type d'objet de bureau) ;
- création ou suppression d'une relation d'inclusion;
- modification d'une des propriétés de l'objet;
- destruction de l'objet.

Toutes ces opérations nous permettent de concevoir n'importe quelle structuration. En voici un exemple :



avec :



/ = relation d'inclusion logique (contenant/contenu)

De tous les exemples possibles de structuration, il ressort les observations suivantes :

-les deux types d'objet Bureau et Document sont obligatoires. En effet, le type Bureau doit être présent car il contient directement, ou indirectement tous les autres types d'objets. Quant au type Document, lui seul permet, grâce à ses occurrences, de stocker de l'information;
- si on ne veut gérer qu'un seul environnement de rangement à la fois, il ne doit exister qu'une seule instance du type Bureau.

Afin de respecter automatiquement ces deux considérations, l'ER que nous fournirons comportera déjà le type Bureau et le type Document En outre, il ne sera pas possible de détruire ces types ni de les créer.Comme une seule occurrence du type Bureau est admise, celle-ci sera déjà créée au sein de l'ER offert.

L'ER automatisé étant représenté sous la forme d'un fichier (la base de données), nous pouvons proposer une dernière opération qui consiste à créer un ER (ce sera une copie d'un fichier).

Par ce dernier apport, toutes les structurations sont possibles. Cependant, elles doivent respecter une hiérarchie dont le sommet est le type Bureau (objet maximal) et où les types Document (objets minimaux) représentent les derniers niveaux.

Pour conclure cette section, retenons que le passage à un ER automatisé a pour effet les résultats suivant:

- la "disparition" de la majorité des types d'objets isolés dansl'ER manuel au profit d'une seule classe, celle des OBJETS RANGEMENT et RANGEABLES.;

- différentes classes (OBJET de BUREAU, Document , ...) ont été complétées par de nouvelles caractéristiques : à la fois des propriétés et des opérations;

- enfin, la création d'une nouvelle classe d'objet (CATEGORIE OBJETS de BUREAU) qui autorise n'importe quelle structuration d'un ER automatisé via les opérations qui lui sont associées.

Un inventaire et une description complète de toutes les opérations que nous avons isolées fera l'objet de la section suivante.

II.3. SPECIFICATIONS FONCTIONNELLES

Nous allons spécifier toutes les opérations disponibles dans l'ER automatisé. Cette description fonctionnelle se fera indépendamment de tout outil d'implémentation; le but final de cette section étant de fixer définitivement les effets des opérations.

II.3.A. PARAMETRES ET SYNTAXE DES OPERATIONS

Cette sous-section se charge de décrire les paramètres des différentes opérations disponibles dans un ER et de définir la syntaxe de ces opérations.

1) Les arguments des opérations

Notons que certains arguments (ID_OBJET par exemple) correspondent à des caractéristiques propres à un système informatique.

* E_R (environnement de rangement)

E_R identifie le nom d'un ER.

* ID_OBJET (identifiant de l'objet)

Tout objet, quel que soit son type, est caractérisé au sein de l'ER par un identifiant interne que nous appellerons ID_OBJET. Cet identifiant interne est le résultat de toute opération d'accès à un objet et il occupe une place d'autant plus importante que son obtention est la condition nécessaire pour pouvoir travailler sur l'objet de bureau qu'il désigne.
Lorsqu'une opération manipulera deux objets, l'objet de rangement sera identifié par ID_CONTENANT tandis que l'objet rangeable sera désigné par ID_CONTENU et, dans un même souci de clarté, un document sera toujours identifié par ID_DOCUMENT.

* TYPE_OBJET (type de l'objet)

Chaque objet présent dans l'ER est caractérisé par son type (TYPE_OBJET). Les valeurs possibles de TYPE_OBJET sont celles qui ont été définies dans la structure de l'ER automatisé

* NOM_OBJET (nom de l'objet de bureau)

Tout objet, quel que soit son type, est caractérisé au sein de l'ER par un second identifiant qui est son nom et que nous appellerons NOM_OBJET. Contrairement à ID_OBJET, NOM_OBJET est un identifiant externe au système dont la valeur est définie à la création de l'objet par son responsable ou son créateur.

* LISTE_PROP (liste des propriétés)

L'ensemble des propriétés d'un objet de l'ER.
Rappelons que tous les objets de bureau sont caractérisés par les propriétés suivantes : un type, un nom, un créateur ou responsable, la date de création, la date de dernière mise à jour, le degré de confidentialité, le mot de passe et le texte libre. Les objets de rangement, en plus des propriétés héritées des objets de bureau, sont caractérisés par leur mode de classement et la propriété sur laquelle se fait la classement. De même, les propriétés des objets rangeables sont : leur localisation d'origine, leur localisation réelle et deux propriétés additionnelles permettant un classement. Les objets de type Document possèdent des caractéristiques qui leur sont propres, à savoir le mode de représentation, le type, le contenu, la référence à l'original, le nom du fichier, le mode de stockage, la nature électronique ou autre, l'origine de la copie et la date de copie. Le lecteur peut se reporter aux deux premières sections de ce chapitre pour connaître la signification exacte de ces propriétés.

* FICHIER

Le corps d'un document dans l'ER se présente toujours, à l'extérieur de celui-ci, sous la forme d'un fichier (dans son acception informatique). Le contenu de ce fichier peut être différent d'un cas à l'autre :
soit il contient l'information proprement dite et alors celle-ci est stockée dans l'ER,
soit il contient le chemin d'accès et le nom du fichier contenant à son tour l'information (ici l'information est de forme électronique mais stockée en dehors de l'ER),
soit il contient la localisation de l'information qui n'est pas de nature électronique et qui se situe en dehors de l'ER.
FICHIER reprend donc le nom d'un fichier externe à l'ER.

* INDICATEUR_LOCALISATION

Nous savons que tout objet rangeable est caractérisé par un lieu de rangement d'origine et une localisation réelle, ces deux lieux pouvant être distincts. INDICATEUR_LOCALISATION permet de déterminer exactement la relation qu'a un objet de rangement avec un de ses composants.

Les valeurs possibles de INDICATEUR_LOCALISATION sont :

0 - l'objet de rangement considéré est à la fois la localisation d'origine et réelle de l'objet rangeable;

1 - l'objet de rangement considéré est la localisation réelle de l'objet rangeable;

2 - l'objet de rangement considéré est le lieu de rangement d'origine de l'objet rangeable.

* INDICATEUR_OPERATION (indicateur de l'opération)

L'utilisateur, après l'exécution d'une opération, dispose d'un indicateur (INDICATEUR_OPERATION) dont chaque valeur est significative de la façon dont l'opération s'est déroulée. En particulier, une valeur nulle (0) de INDICATEUR_OPERATION indique que l'opération s'est exécutée correctement. Par contre, une valeur de INDICATEUR_OPERATION supérieure à 1 implique toujours un abandon de l'exécution de l'opération.
L'ensemble des valeurs possibles de ce code de retour sera repris lors des spécifications externes (cfr chapitre IV).

2) Syntaxe d'une opération

Pour plus de clarté, les paramètres en entrée sont en caractères normaux et les paramètres résultat en caractères gras. Les paramètres ayant les deux types (entrée et sortie) sont repris en italique.

Cela donne :

nom_opération (donnée-entrée-1, donnée-entrée-2, ...
donnée-entsor-1, donnée-entsor-2, ...
donnée-résultat-1, **donnée-résultat-2**, ...),

avec par exemple :

créer (liste_prop, id-contenant,
id-objet, **indicateur_opération**)

II.3.B. LES OPERATIONS APPLICABLES A L'ENVIRONNEMENT DE RANGEMENT

Avant de décrire les opérations une à une, nous allons les présenter.

0) Introduction : Enumération des opérations

a) Opérations d'initialisation d'un environnement de rangement

créer_ER (e_r, **indicateur_opération**);
ouvrir_ER (e_r, **indicateur_opération**);
fermer_ER (e_r, **indicateur_opération**).

b) Opérations de structuration d'un environnement de rangement

créer_nouveau_type (liste_cat, **indicateur_opération**),
supprimer_type_objet (type_objet_cat, **indicateur_opération**),
créer_relation (type_objet_contenant, type_objet_contenu,
indicateur_opération),
supprimer_relation(type_objet_contenant, type_objet_contenu,
indicateur_opération),
modifier_propriétés_type (type_objet_cat, liste_cat,
indicateur_opération),
donner_premier_type (liste_cat, **indicateur_opération**),
donner_type-suivant (*liste_cat*, **indicateur_opération**),
connaître_structure (e_r, fichier, **indicateur_opération**).

c) Opérations d'accès à un objet de bureau :

donner_premier (type_objet, **id_objet**, **indicateur_opération**),
donner_suivant (type_objet, *id_objet*, **indicateur_opération**),
accès_direct (nom_objet, **id_objet**, **indicateur_opération**),
accès_premier (id_contenant, **id_contenu**,
indicateur_localisation, **indicateur_opération**),
accès_suivant (id_contenant, *id_contenu*,
indicateur_localisation, **indicateur_opération**).

d) Opérations de création, de suppression et de modification des objets de bureau:

créer (liste_prop, id_contenant, **id_contenu**,
indicateur_opération),
détruire (id_objet, **indicateur_opération**),
caractéristiques (id_objet, **liste_prop**, **indicateur_opération**),
modification_propriétés(id_objet, liste_prop,
indicateur_opération),
localisation_origine (id_contenu, **id_contenant**,
indicateur_opération),
localisation_réelle (id_contenu, **id_contenant**,
indicateur_opération),
changer_lieu_origine (id_contenu, id_contenant,
indicateur_opération),
déplacer_objet (id_contenu, id_contenant,
indicateur_opération),
réinsérer_place_origine (id_objet, **indicateur_opération**),
..transfert_objet (id_objet, e_r_départ, e_r_arrivée, id_contenant,
id_contenu, **indicateur_opération**).

e) Opérations spécifiques aux documents :

copie (id_document, nom_objet, id_contenant, fichier,**id_contenu**,
indicateur_opération),
obtenir_corps (id_document, fichier, **indicateur_opération**),
remplacer_corps (id_document, fichier, **indicateur_opération**),
placer_corps_hors_ER (id_document, fichier,
indicateur_opération),
placer_corps_in_ER (id_document, fichier,
indicateur_opération),

1) Initialisation d'un environnement de rangement

a) Créer un environnement de rangement

Syntaxe :

créer_ER(e_r, **indicateur_opération**).

Fonction :

crée un environnement de rangement qui sera identifié par "e_r". "**indicateur_opération**" prend la valeur '0' si l'ER a été créé, sinon il a une valeur supérieure. L'ER créé possède déjà deux types d'objets à savoir les types Bureau et Document.

b) Ouvrir un environnement de rangement

Syntaxe :

ouvrir_ER (e_r, **indicateur_opération**)

Fonction :

ouvre l'ER identifié par "e-r". Cette opération a pour objectif de rendre un ER actif afin de permettre toute manipulation ultérieure. "**indicateur_opération**" prend la valeur '0' si l'ER est bien ouvert et la valeur '1' si aucun ER identifié par "e-r" n'a été trouvé.

c) Fermer l'environnement de rangement

Syntaxe :

fermer_ER (e-r, **indicateur_opération**)

Fonction :

ferme l'ER actif s'il y en a un. Une fois fermé plus aucune opération ne sera permise sur celui-ci jusqu'à sa réouverture. Si l'opération ne trouve pas l'ER demandé, "**indicateur_opération**" a la valeur '1'.

2) Opération de structuration d'un environnement de rangement

a) Préambule

Les opérations qui suivent agissent sur la structure d'un ER, elles permettent de créer de nouveaux types d'objet (de les détruire également) ainsi que de définir entre ces différents types des relations d'inclusion.

Ces opérations sont destinées plus particulièrement au programmeur d'application qui aura toute latitude pour créer une hiérarchie spécifique à une application.

Avant d'énoncer les différentes opérations de structuration, il faut noter que deux classes de relation d'inclusion sont interdites : d'une part celle où le type Bureau joue le rôle d'objet rangeable et d'autre part celle où le type Document tient le rôle d'objet de rangement.

* Arguments spécifiques aux opérations de structuration

LISTE_CAT

L'ensemble des propriétés d'une catégorie d'objet de bureau. C'est à dire le type de l'objet, le nombre d'occurrences permises et le nombre d'occurrence présentes.

TYPE_OBJET_CAT

Tout type d'objet présent dans un ER est caractérisé par son nom (TYPE_OBJET_CAT) qui tient le rôle d'identifiant externe.

b) Créer un nouveau type d'objet au sein de l'environnement de rangement

Syntaxe :

créer_nouveau_type (liste_cat, **indicateur_opération**)

Fonction :

crée, dans la structure de l'ER actif, un nouveau type d'objet identifié par "type_objet_cat" présent dans "liste_cat". Si ce type existe déjà, l'opération est abandonnée. La valeur des propriétés de ce nouveau type est disponible dans "liste_cat".
Une valeur spéciale de la propriété "nombre d'occurrences permises" présente dans "liste_cat" indique que l'on ne tient pas compte de ce paramètre lors de la création d'un objet du type considéré. Si l'opération s'est bien déroulée, "**indicateur_opération**" prend la valeur '0', sinon une valeur supérieure.

c) Suppression d'un type d'objet présent dans l'environnement de rangement

Syntaxe :

supprimer_type_objet (type_objet_cat, **indicateur_opération**)

Fonction :

supprime le type d'objet présent dans la structure de l'ER actif, type identifié par "type_objet_cat". Toutes les relations d'inclusion où est impliqué ce type d'objet sont détruites à leur tour. L'opération est interdite pour les types Bureau et Document.

d) Créer une relation d'inclusion entre deux types d'objet

Syntaxe :

créer_relation (type_objet_contenant, type_objet_contenu,
indicateur_opération)
avec type_objet_contenant et type_objet_contenu de type TYPE_OBJET_CAT.

Fonction :

crée une relation d'inclusion entre les deux types d'objet identifiés par "type_objet_contenant" et "type_objet_contenu"; ils seront respectivement considérés comme objet de rangement et objet rangeable : une occurrence du premier pouvant contenir des occurrences du second. Si un ou les deux types identifiés par "type_objet_contenant" et "type_objet_contenu" n'appartiennent pas à la structure de l'ER actif, l'opération est abandonnée. Il en est de même si "type_objet_contenant" identifie un objet de type Document ou si "type_objet_contenu" désigne un objet de type Bureau. Si l'opération s'est bien déroulée, "**indicateur_opération**" prend la valeur '0'; une autre valeur pour le code de retour indique un abandon de l'opération.

e) Supprimer une relation d'inclusion entre deux types d'objet

Syntaxe :

supprimer_relation (type_objet_contenant, type_objet_contenu,
indicateur_opération)
avec type_objet_contenant et type_objet_contenu de type TYPE_OBJET_CAT.

Fonction :

supprime la relation d'inclusion qui existe entre les deux types d'objet identifiés par "type_objet_contenant" et "type_objet_contenu". Si la relation n'existe pas, rien ne se passe. Si les types d'objet identifiés par "type_objet_contenant" et "type_objet_contenu" n'appartiennent pas à la structure de l'ER actif, l'opération est abandonnée.

f) Modifier une ou plusieurs propriétés d'un type d'objet

Syntaxe :

modifier_propriétés_type (type_objet_cat, liste_cat, **indicateur_opération**)

Fonction :

modifie le type d'objet identifié par "type_objet_cat" en remplaçant la valeur de ses propriétés par "liste_cat". Si l'on tente de modifier une propriété non modifiable (cfr remarque), celle-ci reste inchangée. Si le type d'objet considéré n'existe pas dans la structure de l'ER actif, l'opération est abandonnée. Tout incident est signalé par "**indicateur_opération**".

Remarque :

les propriétés modifiables par cette opération sont : le nombre d'occurrences permises et le nombre d'occurrences présentes (avec pour contrainte à respecter que la valeur de la première propriété soit supérieure ou égale à la valeur de la seconde)

g) Donner le premier type d'objet présent dans la structure de l'ER

Syntaxe :

donner_premier_type (**liste_cat**, **indicateur_opération**)

Fonction :

fournit la valeur des propriétés ("**liste_cat**") du premier type d'objet présent dans la structure de l'ER actif. Tout incident lors de l'exécution de l'opération est identifié par la valeur de "**indicateur_opération**".

h) Donner le type d'objet suivant présent dans la structure de l'environnement de rangement

Syntaxe :

donner_type-suivant (*liste_cat*, **indicateur_opération**)

Fonction :

trouve la valeur des propriétés du type d'objet qui, dans la structure de l'ER actif, suit le type d'objet identifié par "type_objet_cat" présent dans "liste_cat"; si ce type existe, la valeur de ses propriétés sera disponible dans "**liste_cat**". Si le type identifié par "type_objet_cat" présent dans "liste_cat" est le dernier de l'ER actif, "**indicateur_opération**" prend la valeur '1' et "**liste_cat**" a une valeur non significative.

i) Connaître la structure d'un environnement de rangement

Syntaxe :

connaître_structure (e_r, fichier, **indicateur_opération**)

Fonction :

donne la structure de l'ER actif sous la forme d'un fichier identifié par "fichier". Pour chaque type d'objet répertorié dans la structure, le fichier reprend ses propriétés ainsi que les relations d'inclusion où il joue le rôle d'objet de rangement et d'objet rangeable.

j) Avertissements :

- lorsqu'il construit une hiérarchie de bureau, le programmeur doit veiller à ce que tout type d'objet soit inclus directement ou indirectement dans le type Bureau;

- la structure d'un bureau peut être modifiée en cours d'exploitation, mais il faut néanmoins le faire avec prudence. En effet, l'ajout de nouveaux types ne pose aucun problème et la suppression d'un type d'objet existant peut se faire à la condition qu'il n'existe plus aucune occurrence de ce type d'objet dans l'ER actif. Parallèlement on peut, sans crainte, ajouter de nouvelles relations d'inclusion. Une relation entre deux types d'objet pourra être détruite si, dans l'ER actif, il n'existe plus aucun objet partageant encore cette relation avec un autre.

3) Accès séquentiel à un objet de bureau de type déterminé

a) Donner la première occurrence d'un objet de bureau de type déterminé

Syntaxe :

donner_premier (type_objet, **id_objet**, **indicateur_opération**)

Fonction :

fournit l'identifiant interne ("**id_objet**") du premier objet de bureau de type 'type_objet' présent dans l'ER actif. L'opération est autorisée pour n'importe quel type d'objet connu de la structure de l'ER actif (du type Bureau au type Document). S'il n'existe aucune occurrence d'objet du type demandé, "**indicateur_opération**" prend la valeur '1' et "**id_objet**" ne désigne aucun objet de bureau.
Tout incident au cours de l'exécution de l'opération est signalé par une valeur de "**indicateur_opération**" supérieure à '1'.

b) Donner l'occurrence suivante d'un objet de bureau de type déterminé

Syntaxe :

donner_suivant (type_objet, *id_objet*, **indicateur_opération**)

Fonction :

trouve l'identifiant interne de l'objet de bureau de type 'type_objet' qui, dans l'ER actif, suit l'objet identifié par "id_objet"; si l'objet considéré existe, son identifiant interne est disponible dans "**id_objet**". L'opération est permise pour des objets de tous types. Si l'objet identifié par "id_objet" n'a pas de suivant, "**indicateur_opération**" prend la valeur '1' et "id_objet" ne désigne aucun objet de bureau.
Un "**indicateur_opération**" d'une valeur supérieure à '1' signale un problème lors de l'opération et un abandon de la recherche.

4) Accès à un objet de bureau par sa valeur d'identifiant externe

Syntaxe :

accès_direct (nom_objet, **id_objet**, **indicateur_opération**)

Fonction :

détermine l'identifiant interne ("**id_objet**") de l'objet de bureau identifié par "nom_objet". La recherche est admise pour n'importe quel type d'objet connu de la structure de l'ER actif. Si, au sein de l'ER, il n'existe aucun objet identifié par "nom_objet", "**indicateur_opération**" prend la valeur '1' et "**id_objet**" ne désigne aucun objet de bureau. Tout problème lors du déroulement de l'opération est identifié par la valeur de "**indicateur_opération**".

5) Parcours séquentiel du contenu d'un objet de rangement

a) Accéder au premier composant d'un objet de rangement

Syntaxe :

accès_premier (id_contenant, **id_contenu**,
indicateur_localisation, **indicateur_opération**)

Fonction :

fournit l'identifiant interne ("**id_contenu**") du premier objet de bureau dont le lieu de localisation est identifié par "id_contenu". L'opération n'a aucun effet si "id_ contenant" identifie des objets de bureau de type Document. Si l'objet désigné par "id_contenant" est vide, "**indicateur_opération**" prend la valeur '1' et "**id_objet**" ne désigne aucun objet rangeable. C'est "**indicateur_localisation**" qui précise la relation d'inclusion exacte qui existe entre les objets identifiés par "id_contenant" et "id_contenu". Une valeur de "**indicateur_opération**" supérieure à '1' dénote un problème lors du déroulement de l'opération.

b) Accéder au composant suivant d'un objet de rangement

Syntaxe :

accès_suivant (id_contenant, *id_contenu*, **indicateur_localisation**, **indicateur_opération**)

Fonction :

détermine l'identifiant interne de l'objet de bureau dont le lieu de localisation est désigné par "id_contenant" et qui suit l'objet identifié par "id_contenu"; si cet objet existe, son identifiant interne est disponible dans "**id_contenu**". L'opération est permise pour des objets de bureau de tous types excepté le type Document. C'est "**indicateur_localisation**" qui précise la relation d'inclusion exacte qui existe entre les objets identifiés par "id_contenant" et "id_contenu". Si l'objet désigné par "id_contenu" n'a pas de suivant, "**indicateur_opération**" prend la valeur '1' et "**id_objet**" ne désigne aucun objet rangeable.
Un problème lors de l'opération se marque par une valeur supérieure à '1' pour "**indicateur_opération**".

6) Création, suppression et modification d'un objet de bureau

a) Créer un objet dans l'environnement de rangement

Syntaxe :

créer (liste_prop, id_contenant, **id_contenu**, **indicateur_opération**)

Fonction :

crée un objet de bureau ("**id_contenu**") de type 'type_objet' présent dans "liste_prop" et l'insère dans l'objet de rangement identifié par "id_contenant". Ce dernier devient à la fois la localisation d'origine et réelle du nouvel objet. Les valeurs des propriétés du futur objet sont disponibles dans "liste_prop". L'opération de création peut porter sur tous les types d'objets à l'exception du type Bureau. Si, dans l'ER actif, il existe déjà un objet de type identique ayant le même nom, l'utilisateur en est averti par "**indicateur_opération**" et la création est abandonnée.
Enfin, cette création n'est pas réalisée si elle entraîne un dépassement du nombre maximum d'objets de ce type permis par la structure de l'ER actif. Si tout s'est bien passé, "**indicateur_opération**" prend la valeur '0'; toute autre valeur suppose un problème lors de la création et l'abandon de l'opération.

b) Détruire un objet appartenant à l'environnement de rangement

Syntaxe :

détruire (id_objet, **indicateur_opération**)

Fonction :

supprime de l'ER actif l'objet identifié par "id_objet". La destruction n'est pas permise pour un objet de type Bureau. De même, toute tentative de destruction d'un objet de rangement non vide est vouée à l'échec.

L'opération entraîne la mise à jour, pour l'objet détruit et s'il y a lieu, du compteur d'occurrence associé au type considéré.
Une valeur de "**indicateur_opération**" supérieure à '0' indique que la destruction n'a pu être menée à terme.

c) Obtenir l'ensemble des propriétés d'un objet de bureau

Syntaxe :

caractéristiques (id_objet, **liste_prop, indicateur_opération**)

Fonction :

fournit les valeurs de toutes les propriétés ("**liste_prop**") de l'objet de bureau identifié par "id_objet". L'opération peut s'appliquer à des objets de n'importe quel type. Une valeur de "**indicateur_opération**" supérieure à 1 indique un abandon de l'opération et, dans ce cas, "**liste_prop**" a une valeur indéterminée.

d) Modifier une ou plusieurs propriétés d'un objet de bureau

Syntaxe :

modification_propriétés (id_objet, liste_prop,
indicateur_opération)

Fonction :

modifie l'objet de bureau identifié par "id_objet" en mettant à jour la valeur de ses propriétés grâce à "liste_prop". L'opération est autorisée pour des objets de bureau de tous types mais pas pour toutes les propriétés. Si l'on tente de modifier des propriétés non modifiables (cfr remarque), ce sont les anciennes valeurs qui sont conservées. Dans le cas où la propriété modifiée est à la base d'un classement, il y a reclassement de l'objet rangeable modifié en vue de tenir compte de la nouvelle valeur de sa propriété.
De même, si la propriété modifiée est 'mode de classement' ou 'propriété de classement', tous les objets de bureau éventuellement inclus dans l'objet de rangement considéré seront reclassés.
Si l'opération s'est bien déroulée, "**indicateur_opération**" prend la valeur '0' sinon il prend une valeur supérieure.

Remarque :

cette opération autorise la modification de la valeur des propriétés communes à tous les objets de bureau c'est-à-dire : créateur ou responsable, confidentialité, mot de passe et des propriétés spécifiques telles que mode de classement (objets de rangement), la propriété de classement (objets de rangement), les deux propriétés additionnelles permettant un classement (objets rangeables), référence à l'original (document), nom du fichier (document) et nature électronique (document).

e) Rechercher la localisation d'un objet de bureau au sein de l'environnement de rangement :

ces deux opérations ne sont pas valides pour des objets de type Bureau.

1) rechercher le lieu de rangement d'origine

Syntaxe :

localisation_origine (id_contenu, **id_contenant**, **indicateur_opération**)

Fonction :

fournit l'identifiant interne ("**id_contenant**") de l'objet de bureau qui, dans l'ER actif, est le lieu de rangement d'origine de l'objet identifié par "id_contenu".

2) rechercher la localisation réelle :

Syntaxe :

localisation_réelle (id_contenu, **id_contenant**,
indicateur_opération)

Fonction :

trouve l'identifiant interne ("**id_contenant**") de l'objet de bureau qui est la localisation réelle de l'objet identifié par "id_contenu".

Un "**indicateur_opération**" avec une valeur supérieure à '0' indique un problème lors de la localisation de l'objet de bureau.

f) Modifier la localisation d'un objet de bureau au sein de l'environnement de rangement:

1) modifier le lieu de rangement d'origine

Syntaxe :

changer_lieu_origine (id_contenu, id_contenant, **indicateur_opération**)

Fonction :

modifie le lieu de rangement d'origine de l'objet de bureau de type quelconque (hormis le type Bureau) identifié par "id_contenu". Le nouveau lieu de rangement est désigné par "id-contenant". La localisation réelle reste, quant à elle, inchangée. Le déplacement n'est effectif que s'il respecte la structure en vigueur dans l'ER actif.

2) modifier la localisation réelle

Syntaxe :

déplacer_objet (id_contenu, id_contenant, **indicateur_opération**)

Fonction :

modifie la localisation réelle de l'objet de bureau identifié par "id_contenu". Le nouveau lieu de rangement est désigné par "id-contenant". L'objet que l'on déplace conserve cependant sa localisation d'origine. Le déplacement n'est effectif que s'il respecte la structure de l'ER actif et si l'objet à déplacer n'est pas de type Bureau.

C'est "**indicateur_opération**" qui désigne la cause d'un éventuel abandon de l'opération.

g) Ranger automatiquement un objet de bureau à sa place d'origine

Syntaxe :

réinsérer_place_origine (id_objet, **indicateur_opération**)

Fonction :

place l'objet de bureau identifié par "id_objet" dans son lieu de rangement d'origine si ce n'était pas déjà le cas. L'opération n'est destinée qu'à des objets rangeables. Après l'application de cette opération à un objet de bureau, la localisation réelle et la localisation d'origine sont confondues. Si l'opération s'est bien déroulée alors "**indicateur_opération**" prend la valeur '0', dans le cas contraire, la cause du problème est explicitée par la valeur de "**indicateur_opération**".

h) Transférer un objet de bureau d'un environnement de rangement à un autre

Syntaxe :

transférer_objet (id_objet, e_r_départ, e_r_destination,
id contenant, **id_contenu**, **indicateur_opération**)

Fonction :

transfert l'objet de bureau identifié par "id_objet" de l'ER désigné par "e_r_départ" vers l'ER indiqué par "e_r_destination". Cet objet ne peut être de type Bureau. Dans son nouveau ER, l'objet transféré est identifié par "**id_contenu**" et "id_contenant" désigne son lieu de rangement (d'origine et réel). Si l'opération s'est bien déroulée alors "**indicateur_opération**" prend la valeur '0', dans le cas contraire, l'origine du problème est explicitée par la valeur de "**indicateur_opération**".

7) Opérations spécifiques aux documents

a) Dupliquer un objet de bureau de type Document

Syntaxe :

copie (id_document, nom_objet, id_contenant, fichier, **id_contenu**, **indicateur_opération**)

Fonction :

crée un nouveau document ("**id_contenu**") à partir du document identifié par "id_document". Le nouveau document possède les mêmes valeurs de propriété que l'original sauf pour les propriétés nom, nom de fichier ("fichier"), l'origine de la copie et la date de copie, et il est inséré dans l'objet de rangement identifié par "id_contenant". Si l'opération s'est bien déroulée, "**indicateur_opération**" prend la valeur '0' sinon une valeur supérieure.

b) Fournir le corps d'un objet de bureau de type Document

Syntaxe :

obtenir_corps (id_document, fichier, **indicateur_opération**)

Fonction :

fournit le corps du document identifié par "id_document" sous la forme du fichier externe désigné par "fichier". L'opération se charge de vérifier s'il n'existe pas de fichier déjà identifié par "fichier" : si c'est le cas, "**indicateur_opération**" signale le problème et l'opération est abandonnée.

c) Modifier le corps d'un document

Syntaxe :

remplacer_corps (id_document, fichier, **indicateur_opération**)

Fonction :

modifie le corps de l'objet de bureau de type Document identifié par "id_document" en le remplaçant par le contenu du fichier désigné par "fichier". Le fichier identifié par "fichier" est détruit. Cette opération est unique quel que soit le mode de stockage du document (dans ou en dehors de l'ER) et/ou sa nature (électronique ou autre). Une valeur de "**indicateur_opération**" supérieure à '1' dénote un problème lors du déroulement de l'opération.

d) Modifier le mode de stockage d'un document :

1) passage à un stockage en dehors de l'environnement de rangement

Syntaxe :

placer_corps_hors_ER (id_document, fichier,
indicateur_opération)

Fonction :

modifie le mode de stockage du corps du document identifié par "id_document" en le sortant de l'ER actif. L'opération exige que l'information attachée au document soit stockée dans l'ER. Après cette opération, la propriété 'mode de stockage' prend la valeur 'non'. Enfin, il faut noter que l'opération se charge de faire une copie externe du corps du document : le fichier externe est identifié par "fichier". Si "fichier" désigne déjà un fichier externe, l'opération est abandonnée.

2) passage à un stockage dans l'environnement de rangement

Syntaxe :

placer_corps_in_ER (id_document, fichier, **indicateur_opération**)

Fonction :

modifie le mode de stockage du corps du document identifié par "id_document" en chargeant dans l'ER actif le contenu du fichier désigné par "fichier" Suite à cette opération, l'information attachée au document considéré est présente dans l'ER actif (la propriété 'mode de stockage' a la valeur 'vrai'). Enfin, il faut noter que l'opération se charge de détruire le fichier externe identifié par "fichier"; ceci afin de maintenir le principe d'unicité de l'information.

Chapitre III L'ARCHITECTURE DES DONNEES

Nous avons réalisé une étude de l'existant (l'ER manuel) et examiné le passage d'un ER manuel à un ER automatisé en utilisant une approche orientée objet. C'est ainsi que nous avons isolé des objets (les objets de bureau et les objets catégorie objets de bureau), précisé leurs propriétés (étude d'un ER manuel et passage à un ER automatisé) ainsi que les opérations qui leur sont associées (les spécifications fonctionnelles). Des classes d'objets et des sous-classes ont été distinguées. Enfin, il a été fourni un exemple de hiérarchisation possible au sein d'un ER automatisé.

Les outils mis à notre disposition (NDBS et le langage de programmation PASCAL) ne nous permettent pas d'implémenter directement en termes d'OBJET. En effet, comme nous le verrons, NDBS propose un modèle de description des données basé non sur le concept d'OBJET mais sur celui d'ENTITE (modèle Entité/Association). De plus, le langage PASCAL utilisé n'est pas un langage orienté objet.

Nous devons donc effectuer une traduction de la modélisation exprimée par le concept OBJET en une modélisation basée sur le concept ENTITE, tout en gardant l'acquis obtenu grâce à l'approche orientée objet. Afin de réaliser cette traduction, nous conservons de l'approche orientée objet la notion fondamentale de **type abstrait de données**, de sorte que toutes les opérations définies avec l'approche orientée objet restent de mise et que le problème se "résume" à concevoir une représentation du type abstrait "objet de bureau" et du type abstrait "catégorie objets de bureau" en terme des concepts du modèle E/A.

Pour aider à comprendre comment cette traduction a été réalisée, le modèle E/A sera brièvement rappelé et une présentation de NDBS sera faite. Ensuite, une représentation des objets de bureau et des objets catégorie objets de bureau suivant l'approche Entité/Association sera présentée et commentée. Nous établirons ensuite la liste de tous les attributs de ces objets.
Enfin, nous présentons les étapes de l'élaboration d'un schéma conforme au SGBD NDBS.

III.1. LE MODELE ENTITÉ/ASSOCIATION : RAPPEL

Le modèle Entité/Association [BODART, PIGNEUR;1983] est un modèle conceptuel de structuration des informations.

Rappelons brièvement que, dans ce modèle, tout objet ou concept du réel (par exemple une personne, un examen,...) est représenté par une entité. Chaque entité appartient à une classe appelée Type d'entité (ex : le Type d'Entité PERSONNE). Quant aux diverses relations possibles entre des entités distinctes (ex : Dupont présente l'examen de base de données), elles sont représentées par des associations qui sont classées en type d'association (ex : PRESENTATION D'EXAMEN). Chaque entité joue un rôle dans une association . Le nombre d'entités en nombre minimum et en nombre maximum participant à un rôle est appelé contrainte de connectivité. Enfin, les propriétés des objets du réel sont représentées par des attributs qui sont ajoutés aux entités et aux associations.

Pour un type d'entité ou un type d'association, on parlera en termes d'attribut tandis que pour une entité ou une association on exprimera des valeurs d'attribut.

Certaines propriétés ne sont pas modélisables . Elles sont dès lors représentées par un certain nombre de contraintes d'intégrité.

Pour les définitions précises de tous les concepts du modèle, nous renvoyons le lecteur à [BODART,PIGNEUR;1983]. Nous nous attachons cependant à définir les propriétés d'un attribut.

Un attribut d'un type d'entité ou d'un type d'association est **identifiant** si chacune de ses valeurs désigne, de manière univoque, une entité du type d'entité ou une association du type d'association.
L'identifiant constitue une contrainte d'intégrité dans le modèle Entité/Association (exemple : tout objet de bureau est identifié soit par son nom soit par un code).

Un attribut est **simple** si, pour chaque entité ou association à laquelle il est rattaché, il ne peut prendre qu'une et une seule valeur. Inversément, un attribut est **répétitif** lorsqu'il peut prendre plusieurs valeurs pour une même entité ou association.
Exemple d'attribut simple : tout objet de bureau n'est que d'un et un seul t ype ("TIROIR",...).

Un attribut est **décomposable** si on peut le décomposer en fragments qui restent toujours porteurs d'une signification.
Exemple : l'attribut date-création d'un objet de bureau est décomposable en trois autres attributs :

- jour-création ,
- mois-création,
- année-création .

Un attribut **élémentaire** est un attribut dont chaque valeur est atomique. Toute décomposition d'un attribut élémentaire entraîne une perte de signification sémantique.

Un attribut est **obligatoire** si, à toute entité ou association que précise cet attribut, doit être attachée une valeur de l'attribut.
Exemple : à tout objet de bureau doit être associée une date de création.

Un attribut **facultatif** exprime qu'à une entité ou une association ne doit pas nécessairement être associée une valeur de l'attribut.
Exemple : on peut associer si on le désire un mot-clé à une armoire.

Tout attribut que l'on déclare, qu'il soit obligatoire ou facultatif, doit recevoir une valeur. Dès lors, un attribut facultatif non précisé (n'ayant pas reçu de valeurs par l'utilisateur) prend une valeur dite "inexistante".
Exemple : pour l'attribut mot-clé d'un objet de bureau, la valeur inexistante est une chaîne de caractères vide.

Les propriétés que nous venons de décrire sont celles qui sont couramment utilisées [BODART,HAINAUT]. Le contexte dans lequel nous travaillons (élaboration d'une application informatique) permet l'ajout des propriétés décrites ci-dessous.

Un attribut est **définitif** si lors de la création de l'entité ou de l'association à laquelle il est rattaché, il reçoit une valeur qui ne peut plus être modifiée.
Exemple : lors de la création d'un tiroir, son type est définitif.

Un attribut est **modifiable** si sa ou ses valeurs peuvent être modifiées au cours de l'application.

Un attribut est dit **automatique** si les diverses valeurs qu'il peut prendre peuvent être déterminées automatiquement, c'est-à-dire, sans intervention externe.
Exemple : la date de dernière mise à jour d'un objet de bureau peut être générée automatiquement lors d'une opération impliquant une modification de son contenu.

Un attribut **manuel** est un attribut dont les valeurs ne peuvent être données que par l'utilisateur de l'application.
Exemple : l'aspect confidentiel d'un objet de bureau ne peut être précisé que par son créateur.

Les types de valeur des attributs que nous reprenons sont ceux prédéfinis au sein de la majorité des S.G.B.D et des langages de programmation, à savoir :

- le type chaîne de caractères,
  exemple : l'attribut créateur-responsable d'un objet de bureau,
- le type numérique entier ;
  exemple : le jour de la création d'un objet ;
- le type booléen qui comporte deux valeurs : vrai ou faux ;
  exemple : l'attribut confidentialité prend la valeur vrai si
  l'objet qu'il précise est confidentiel, la valeur faux si non.

Ces types prédéfinis peuvent être combinés pour créer de nouveaux types.

Exemple : le type date se compose des types entier (jour,mois,année).

III.2. PRESENTATION DE N.D.B.S

III.2.A. INTRODUCTION

Cette section constitue un résumé du document intitulé "NDBS : A simple data base system for small computers" [Hainaut,1987].
Elle présente le Système de Gestion de Base de Données sur lequel repose nos primitives et aide, par la même, le lecteur à comprendre les choix que nous avons du faire.

N.D.B.S. (Network Data Base System) est un système de gestion de base de données de type réseau permettant de développer rapidement des applications en Pascal sur des micro-ordinateurs.

III.2.B. L'ENVIRONNEMENT DE N.D.B.S.

L'environnement de N.D.B.S. se compose de trois éléments à savoir
- un ensemble de procédures (**Data Base Handler**) utilisables par des programmes Pascal;
- un dictionnaire de données et un gestionnaire de schéma;
- un langage de haut niveau (**ADL-Pascal**).

Nous reviendrons plus tard sur les différentes procédures offertes au programmeur. Notons simplement que seulement douze opérations de base sont nécessaires à la création de programmes et qu'elles sont compatibles avec n'importe quel programme Pascal.

Le gestionnaire de schéma permet à l'utilisateur de décrire une base de données et de la modifier. Le modèle des données (c'est-à-dire la façon dont les données peuvent être structurées dans la base de données) est dérivé de l'approche **Entité/Association**.
Au sein de N.D.B.S., la distinction a été faite entre la structure sémantique des données d'une part et leurs paramètres techniques d'autre part; l'avantage étant que tout programmeur peut ignorer les paramètres techniques sans que cela ne porte à conséquence.

Enfin, le langage de haut niveau (ADL-Pascal) permet, entre autres, au programmeur d'écrire des programmes plus concis que ceux écrits avec les procédures du DBH.

III.2.C. CARACTÉRISTIQUES DE N.D.B.S.

Un schéma N.D.B.S. se charge de décrire la base de données, le type et les attributs des entités ainsi que les identifiants et les types de relation entre les types d'entité.

1) Composants logiques d'un schéma N.D.B.S. :

- une **base de données** dont le nom correspond au nom d'un fichier (un chaîne de 1 à 8 caractères, sans extension);

- un **type d'entité** portant un nom qui est un identificateur Pascal. (notons, qu'au sein d'un schéma, deux types d'entité ne peuvent porter le même nom);

- un **attribut d'entité** ayant également un nom qui est un identificateur Pascal. Pour un même type d'entité, les attributs doivent avoir des noms différents et seul un d'entre eux peut en être l'identifiant. Un type d'entité peut ne pas avoir d'attributs. Le système admet les types suivant pour les attributs : entiers, caractères, booléen, réels et chaînes de caractères;

- un **type de relation**, portant un nom qui est un identificateur Pascal,.est défini entre deux types d'entité non nécessairement distincts.

2) Composants physiques d'un schéma N.D.B.S. :

- une entité est représentée sous la forme d'une chaîne d'octets appelée **enregistrement d'entité** (entity record);

- les données de la base sont stockées dans un fichier qui est divisé en **pages**. Un enregistrement (une entité) est rangé dans une des pages du fichier sans jamais s'étendre sur plusieurs d'entre elles. La taille d'une page est, par défaut, de 1024 octets.

- le **page range** c'est-à-dire l'intervalle de pages dans lequel sont stockés les enregistrements d'un type d'entité.

- le **plan de rangement** (storage scheme) c'est-à-dire la façon dont les enregistrements sont stockés dans les pages; plan qui est spécifique à chaque type d'entité. N.D.B.S. offre deux modes de rangement :
soit le **random storage scheme** qui consiste à ranger un enregistrement au hasard dans une des pages et cela le plus uniformément possible;

soit le **clustered storage scheme** où la stratégie suivie est de stocker un enregistrement dans la page à laquelle on a accédé en dernier lieu, cette façon de faire tend à regrouper sur des pages contiguës les entités créées simultanément. Dès lors, un accès séquentiel à ces entités sera très efficace du point de vue des performances en opérations d'entrée-sortie

- le **tampon** (buffer) qui est une zone de la mémoire centrale dans laquelle N.D.B.S. stocke les pages du fichier de données. Quand le programme a besoin d'un enregistrement d'entité, le système de gestion va d'abord voir si celui-ci n'est pas présent dans le tampon; si ce n'est pas le cas, la page correspondant à l'enregistrement est stockée à son tour dans le buffer. Le choix judicieux de la taille du buffer a pour effet d'augmenter l'efficacité des programmes.

- l'**index** qui permet d'accéder plus rapidement à l'entité dont on a donné la valeur d'identifiant.

Parmi les six composants physiques qui ont été décrits, certains peuvent être manipulés en vue d'optimiser les programmes.
Les composants dont on peut modifier la valeur sont : la taille des pages et du tampon, le champ des pages et le plan de stockage de chaque type d'entité.

III.2.D. LES PROCEDURES DE N.D.B.S. (Data Base Handler)

1) Introduction

N.D.B.S. offre au programmeur un ensemble de procédures Pascal qui permettent :

- d'ouvrir et de fermer une base de données;
- de balayer séquentiellement les entités d'un type déterminé;
- d'obtenir les entités reliées à d'autres via un chemin (une relation);
- d'accéder à une entité d'une valeur d'identifiant donné;
- de créer, modifier et détruire des entités;
- de relier et déconnecter des entités.

2) Les nouveaux types de données et les variables

Pour pouvoir désigner des entités et transmettre les valeurs d'attributs, de nouveaux types de données et des variables sont mis à la disposition du programmeur. Ce sont :

- le type **référence** dont une valeur peut désigner une entité dans la base de données (le type référence est appelé **DBREF**);

- la **variable de référence** de type DBREF. Les variables de référence peuvent être organisées en tableau ou en enregistrement mais pas en fichier ni en ensemble. En outre, elles ne peuvent être manipulées que par les procédures du DBH et ne peuvent être ni **lues**, ni **écrites**;

- la **variable entité** contenant à la fois la référence d'une entité et les valeurs des attributs de cette entité;

- le nom d'une base de données;

- le nom d'un type d'entité;

- le nom d'un type de relation entre deux types d'entité;

- et enfin le code de retour du DBH, variable globale de type entier appelée **DBSTATUS.** Il indique comment l'opération s'est déroulée.

3) Les arguments des procédures

Pour permettre au lecteur de comprendre les spécifications des procédures du DBH, nous présentons ci-dessous leurs arguments.

- **data-base** :
  type : string[64];
  signification : le nom d'une base de données.

- **entity-type** :
  type : entier;
  signification : la valeur entière identifie un type d'entité valide de la base de données active.

- **path-type** :
  type : entier;
  signification : la valeur entière identifie un type de relation valide de la base de données active; si la valeur est positive, elle désigne le type de chemin 1-N; si la valeur est négative, le type de chemin indiqué est le type de chemin inverse N-1.

- **ent-var** :
  type : T <nom du type d'entité>;
  signification : une variable entité.

- **targ-ent-var** :
  type : T <type d'entité>;
  signification : une variable entité désignant une entité cible d'un chemin.

- **orig-ent-var** :
  type : T <type d'entité>;
  signification : une variable entité désignant une entité origine d'un chemin.

- **ref-var** :
  type : DBREF;
  signification : une variable référence.

- **var**
  type : soit DBREF, soit une variable entité.

4) Spécification des procédures

a) Ouvrir et fermer une base de données :

* **dbopen** (data-base)
Fonction : ouvrir la base de données appelée "data-base" si elle existe et la rendre disponible pour le programmeur. Elle devient la base de données active du programme;

* **dbclose**
Fonction : fermer la base de donnée active.

b) Parcours séquentiel des entités d'un type donné :

* **dbfirst** (entity-type, ent-var)
Fonction : trouver la première entité du type "entity-type" dans la base de données active et stocker sa référence et ses valeurs d'attributs dans la variable "ent-var";

* **dbnext** (entity-type, ent-var)
Fonction : trouver l'entité de type "entity-type" qui suit celle désignée par "ent-var" dans la base de données active et stocker sa référence et ses valeurs d'attributs dans la variable "ent-var".

c) Obtenir une entité par sa valeur d'identifiant :

* **dbid** (entity-type, ent-var)
Fonction : trouver dans la base de données active l'entité du type "entity-type" qui est identifiée par sa valeur d'identifiant préalablement stockée dans "ent-var" et stocker sa référence et ses valeurs d'attributs dans la variable "ent-var".

d) Accès direct à une entité :

* **dbdirect** (entity-type, ent-var, var)
Fonction : trouver dans la base de données active l'entité de type "entity-type" référencée par la variable entité ou la variable référence "ref-var" et stocker sa référence et ses valeurs d'attributs dans la variable "ent-var".

e) Parcours d'un chemin d'accès :

* **dbfpath** (targ-ent-var, orig-ent-var, path-type).
Fonction : trouver la première entité reliée à l'entité "orig-ent-var" par le chemin "path-type" dans la base de données active et stocker sa référence et ses valeurs d'attributs dans la variable "targ-ent-var". Cette primitive peut être utilisée à la fois pour les chemins 1-N et N-1;

* **dbnpath** (targ-ent-var, orig-ent-var, path-type)
Fonction : trouver l'entité qui suit l'entité "targ-ent-var" parmi celles connectées à l'entité "orig-ent-var" par le chemin "path-type" dans la base de données active et stocker sa référence et ses valeurs d'attributs dans la variable "targ- ent-var";

* **dbtestpath** (targ-ent-var, orig-ent-var, path-type): boolean
Fonction : retourner la valeur 'vrai' si les entités "targ-ent-var" et "orig-ent-var" sont reliées par "path-type" ou retourner la valeur 'faux' dans le cas contraire.

f) Création, destruction et mise à jour d'une entité :

* **dbcreate** (entity-type, ent-var)
Fonction : insérer dans la base de données une entité du type "entity- type". Les valeurs d'attributs viennent de "ent-var" et stocker la référence de la nouvelle entité dans la variable "ent-var";

* **dbdelete** (entity-type, ent-var)
Fonction : effacer de la base de données active l'entité désignée par "ent-var" de type "entity-type". Si l'entité à détruire est une cible dans des chemins 1-N, elle sera d'abord retirée de ceux-ci. Si l'entité à détruire est une origine dans des chemins 1-N, leurs entités cibles seront d'abord enlevées de ces chemins et seront donc à nouveau libres;

* **dbmodify** (entity-type, ent-var)
Fonction : modifier l'entité du type "entity-type" désignée par "ent-var". Les nouvelles valeurs d'attributs sont stockées dans "ent-var".

g) Manipulation des chemins d'accès :

* **dbinsert** (targ-ent-var, orig-ent-var, path-type)
Fonction : relier l'entité "targ-ent-var" à l'entité "orig-ent-var" par une occurrence du chemin 1-N de type "path-type".

* **dbremove** (targ-ent-var, orig-ent-var, path-type)
Fonction : déconnecter l'entité "targ-ent-var" de l'entité "orig-ent-var".

h) Fonctions de N.D.B.S.

En plus des procédures énoncées, N.D.B.S. fournit des fonctions qui permettent de tester la valeur du code retour DBSTATUS. Elles sont au nombre de cinq : dbfound, dbnotfound, dbnonunique, dbsevere et dbpanic. Le lecteur intéressé se reportera au document original pour en savoir plus.

i) Manipulation des variables entité et référence

La variable référence et la variable entité étant deux nouveaux concepts, N.D.B.S. offre des procédures particulières pour pouvoir traiter ces variables.

Brièvement, ces procédures permettent de copier des variables entité (dbcopyatt, dbcopyall et dbcopyref), de les comparer (dbequal) et enfin de manipuler les variables de référence (dbclear et dbnull).

j) Intégrité de la base de données :

*** dbcommit**
Fonction : écrire dans le fichier de données les dernières modifications de la base de données. En cas d'accident, celles-ci ne seront pas perdues.

CONCLUSION

N.D.B.S. constitue un système de gestion de base de données complet qui fournit tous les éléments nécessaires pour élaborer simplement et efficacement des applications de n'importe quelle complexité.

Nous nous sommes volontairement attardés sur les procédures du DBH car toutes nos primitives les utilisent. Ce sont ces procédures qui nous permettent de travailler sur le représentation des types abstraits définis, représentation qui est l'objet de la section suivante.

III.3. LE SCHÉMA ENTITÉ/ASSOCIATION

Il nous faut maintenant donner la représentation des types abstraits OBJETS de BUREAU et CATEGORIE OBJETS de BUREAU (définis dans l'analyse fonctionnelle) en terme des concepts du modèle E/A.

Nous allons donc commenter la traduction de la modélisation construite avec le concept OBJET en une modélisation exprimée suivant les concepts du modèle E/A. Les commentaires qui suivent se rapportent au schéma E/A présenté plus loin page

Premièrement, un objet de la classe CATEGORIE OBJETS de BUREAU est décrit au moyen d'une entité de type CATEGORIE OBJET. Les propriétés de la classe deviennent les attributs du type d'entité CATEGORIE OBJET. Ces attributs sont les suivants : TYPE_OBJET_CAT (nom d'un type d'objet), NB_OCCU_PERMISES et NB_OCCU_PRESENTES. Enfin, les relations entre les objets de cette classe sont concrétisées par un type d'association INCLUSION.

Nous avons ainsi modélisé la structuration d'un ER automatisé en terme du modèle E/A.

Deuxièmement,un objet de bureau est représenté au moyen d'une entité de type OBJET. Les différentes propriétés de la classe OBJETS de BUREAU deviennent les attributs du type d'entité OBJET. Les propriétés de la sous-classe des objets rangeables sont matérialisées au moyen de deux types d'association récursives (LOCALISATION et LOCALISATION D'ORIGINE).

L'unique objet de type Bureau ne possède pas ces deux propriétés puisqu'il n'appartient qu'à la seule classe des objets de rangement. Aussi, l'entité dont l'attribut TYPE-OBJET a pour valeur 'BUREAU' ne participe pas aux rôles "est localisé dans " et "a pour localisation d'origine".

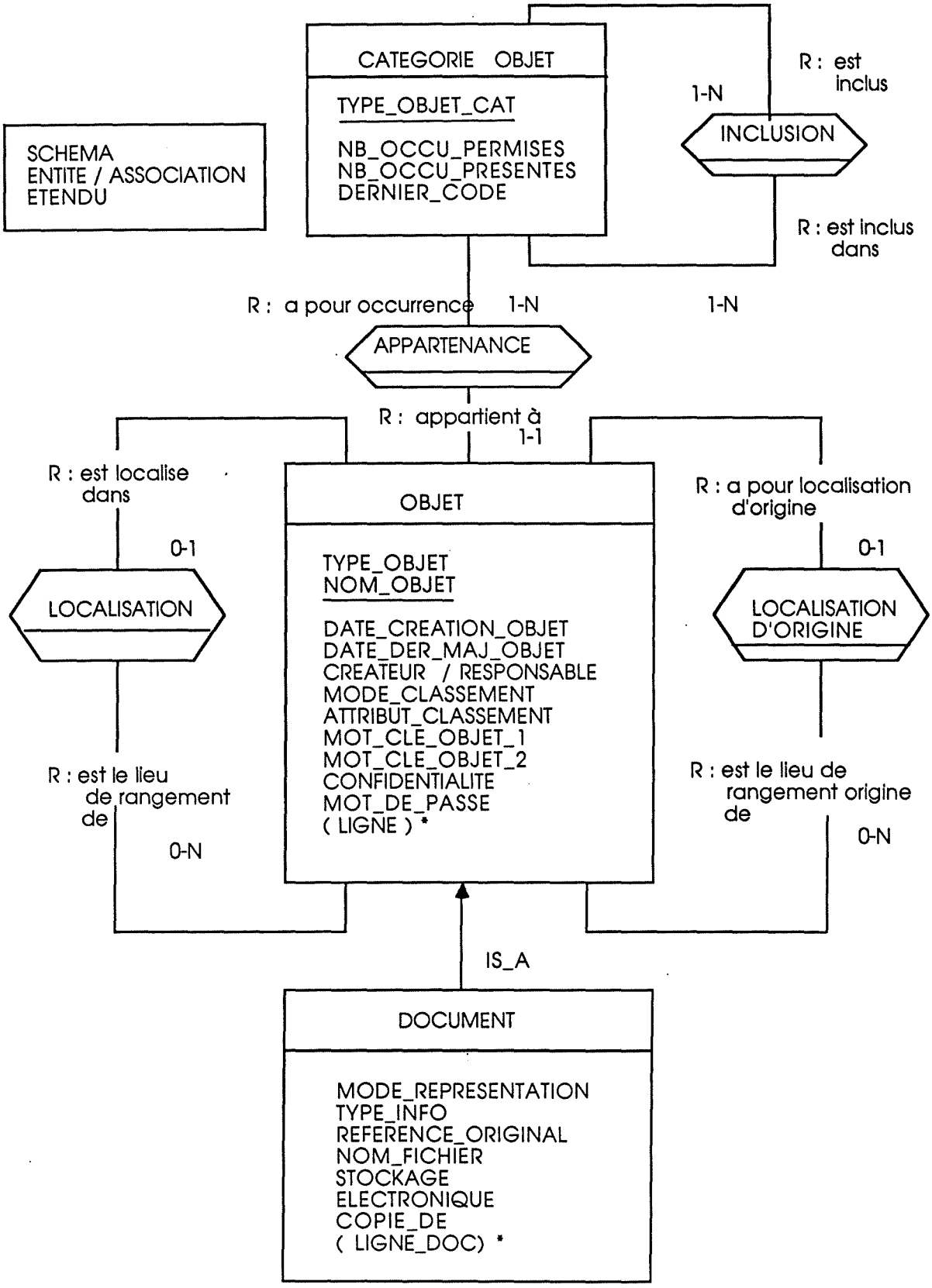
De même, le type Document n'appartenant qu'à la sous-classe des objets rangeables, toutes les entités dont la valeur d'attribut TYPE-OBJET est 'DOCUMENT' ne participent pas aux rôles "est le lieu de rangement de " et "est le lieu de rangement d'origine de ".

Le type Document possède en plus des propriétés de la classe OBJETS de BUREAU des propriétés spécifiques. Pour sa représentation, nous avons utilisé une structure sémantique avancée [HAINAUT,1987] : la généralisation/spécialisation. Un document est dès lors décrit par une entité de type OBJET (c'est un objet de bureau comme tous les autres) et par une entité de type DOCUMENT où les propriétés spécifiques à un document sont reprises en tant qu'attributs. Les deux types d'entité sont reliées par une relation IS_A exprimant que le document est un objet de bureau et qu'il en hérite des propriétés.

Tous les objets de bureau sont donc maintenant représentés.

Le lien qui existe entre les éléments de la structure d'un ER (les types d'objet Armoire, Tiroir,...) et les occurrences de ces éléments est matérialisé par un type d'association APPARTENANCE. Le lien entre les deux classes est ainsi représenté.
Pour des raisons d'efficacité nous avons introduit une redondance à cette association d'appartenance sous la forme d'un attribut TYPE_OBJET, attribut ajouté au type d'entité OBJET.

L'utilisation de structures sémantiques avancées justifie le terme de schéma Entité/Association Etendu [HAINAUT,1987].


SCHEMA
ENTITE / ASSOCIATION
ETENDU
CATEGORIE OBJET
TYPE_OBJET_CAT
NB_OCCU_PERMISES
NB_OCCU_PRESENTES
DERNIER_CODE
R : est inclus
1-N
INCLUSION
R : est inclus dans
R : a pour occurrence
1-N
1-N
APPARTENANCE
R : appartient à
1-1
R : est localise dans
0-1
LOCALISATION
R : est le lieu de rangement de
0-N
OBJET
TYPE_OBJET
NOM_OBJET
DATE_CREATION_OBJET
DATE_DER_MAJ_OBJET
CREATEUR / RESPONSABLE
MODE_CLASSEMENT
ATTRIBUT_CLASSEMENT
MOT_CLE_OBJET_1
MOT_CLE_OBJET_2
CONFIDENTIALITE
MOT_DE_PASSE
(LIGNE) *
R : a pour localisation d'origine
0-1
LOCALISATION D'ORIGINE
R : est le lieu de rangement origine de
0-N
IS_A
DOCUMENT
MODE_REPRESENTATION
TYPE_INFO
REFERENCE_ORIGINAL
NOM_FICHIER
STOCKAGE
ELECTRONIQUE
COPIE_DE
(LIGNE_DOC) *

CONTRAINTES D'INTEGRITE SUPPLEMENTAIRES :

* Toute occurrence d'objet (sauf l'occurrence bureau) participe aux rôles "est localisé dans " et "a pour localisation d'origine".

  Toute occurrence d'objet dont TYPE-OBJET = 'DOCUMENT' ne participe pas aux rôles "est le lieu de rangement" et "est le lieu de rangement d'origine de ".

* Les domaines de valeurs :
  - TYPE_OBJET_CAT et TYPE_OBJET :
    les valeurs "BUREAU" et "DOCUMENT" sont obligatoires.

  - MODE_CLASSEMENT_OBJET =
    { 1 (non trié), 2 (trié croissant), 3 (trié décroissant)}.

  - ATTRIBUT_CLASSEMENT_OBJET =
    {1 (type_objet), 2 (nom_objet), 3 (date_création),
    4 (créateur / responsable), 5 (date_der_maj),
    6 (mot_clé_objet_1), 7 (mot_clé_objet_2) }

  - MODE_REPRESENTATION = { graphique, écrit, oral,...}.

  - TYPE_INFO = { lettre, formulaire,....}...

* Contraintes sur attributs :
  - pour toute occurrence d'OBJET :
    DATE_DER_MAJ >= DATE_CREATION_OBJET

  - pour toute occurrence d'OBJET et pour toute occurrence :
    DOCUMENT associée :
    DOCUMENT.DATE_COPIE >= OBJET.DATE_CREATION_OBJET.

  - pour toute occurrence de CATEGORIE-OBJET :
    NB_OCCU_PRESENTES ≤= NB_OCCU_PERMISES.

* Contrainte référentielle :
  OBJET.TYPE_OBJET ⊆ CATEGORIE_OBJET.TYPE_OBJET_CAT.

III.4. LES ATTRIBUTS

Nous pouvons à présent préciser les différentes propriétés des attributs. Nous donnerons d'abord les attributs communs aux objets de bureau et, ensuite, les attributs plus spécifiques aux documents.

Enfin, les propriétés des attributs relatifs aux objets composant la structuration du bureau seront précisées.

III.4.A POUR LES OBJETS DE BUREAU

. type_objet

Attribut donnant le type de l'objet de bureau.
Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
définitif,
manuel,
type de valeur : type prédéfini chaîne de caractères,
exemples de valeur : (bureau, armoire, tiroir, plan de travail, pile, poubelle, chemise ,document).

. nom_objet

Par défaut, le nom de l'objet sera généré automatiquement sous la forme d'un code qui se compose des trois premières lettres de type de l'objet et de la valeur d'un compteur (incrémenté à chaque création d'un nouvel objet du même type). Si on désire donner un nom à l'objet, les trois premières lettres du type seront automatiquement concaténées en début d'attribut.

Attribut identifiant,
simple,
élémentaire,
obligatoire,
modifiable,
manuel,
type de valeur : type prédéfini chaîne de caractères.

. date_création_objet

Attribut non-identifiant,
simple,
décomposable,
obligatoire,
définitif,
automatique,
type de valeur : type date,
décomposable en jour (type entier),
mois (type entier),
année (type entier).

. créateur/responsable_objet

Reprend le nom du créateur ou du responsable de l'objet.
Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
modifiable,
manuel,
type de valeur : type prédéfini chaîne de caractères.

. date_der_maj_objet

La valeur de cet attribut sera automatiquement mise à jour lors d'une modification du contenu de l'objet, qu'il soit objet de rangement ou document.

Attribut non-identifiant,
simple,
décomposable,
obligatoire,
automatique,
modifiable (par le système),
type de valeur : type date.

. **mode_de_classement_objet**

Caractérise l'ordre de rangement des composants d'un objet de rangement.

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini entier
domaine de valeur : 0 : non trié (= ordre d'insertion);
1 : trié croissant;
2 : trié décroissant;

attribut_classement

Désigne sur quel attribut de l'objet rangé (objet contenu dans l'objet considéré) se fera le classement. Cette désignation se fera sur une valeur numérique qui correspond à l'ordre de présentation des attributs.

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini entier.

. **mot_clé _objet_1**

Cet attribut (de type alphanumérique) offre la possibilité de donner une autre caractéristique à l'objet , caractéristique qui pourra être à la base d'un classement.

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères.

. mot_clé_objet_2

Même utilité que l'attribut mot_clé_objet_1 mais est de type numérique.

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
manuel,
modifiable,
type de valeur : type prédéfini entier.

. ligne_objet

Cet attribut permet une description de l'objet par un texte.

Attribut non-identifiant,
répétitif,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères.

. confidentialité

Détermine si l'objet est confidentiel ou non.

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini booléen.

. mot_de_passe

Correspond à la clé qui permettra d'accéder au contenu de l'objet.

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères

III.4.B POUR LE DOCUMENT,LES ATTRIBUTS SUIVANT VIENNENT EN AJOUT:

. mode de représentation

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères,
domaine de valeurs : (oral, écrit, graphique,...).

. type_info

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
manuel,
définitif,
type de valeur : type prédéfini chaîne de caractères, exemples de valeurs : (message, formulaire, entretien, mémo,...).

. référence original

Cet attribut fournit la référence à l'original du document s'il existe (ex : extrait d'un tel livre, ...).

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
manuel,
définitif
type de valeur : type prédéfini chaîne de caractères.

. nom_fichier

Désigne le nom du fichier sous-lequel le document proprement-dit a été inséré dans l'environnement de rangement et sous-lequel il sera restitué.

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères.

. stockage

Cet attribut précise si le corps du document est stocké ou non dans la base de données.

Attribut non-identifiant,
simple,
élémentaire,
obligatoire,
modifiable (par le système uniquement),
manuel,
type de valeur : type prédéfini booléen.

. copie_de

Cet attribut donne le nom du document qui a été à l'origine de la copie.

Attribut non-identifiant,
simple,
élémentaire,
facultatif,
automatique,
définitif,
type de valeur : type prédéfini chaîne de caractères.

. date_copie

Attribut qui précise la date à laquelle le document a été copié.

Attribut non-identifiant,
simple,
décomposable,
facultatif,
automatique,
définitif,
type de valeur : type date.

. électronique
Attribut permettant de distinguer les documents électroniques des documents non électroniques .

Attribut non identifiant,
simple,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini booléen.

. ligne_doc

Cet attribut contient le corps du document.

Attribut non-identifiant,
répétitif,
élémentaire,
obligatoire,
manuel,
modifiable,
type de valeur : type prédéfini chaîne de caractères.

4.C. POUR LES DIFFÉRENTS OBJETS COMPOSANT LA STRUCTURE DE L'ENVIRONNEMENT DE RANGEMENT ET IMPLIQUANT LA HIÉRARCHISATION :

. type_objet_cat

Attribut dont les valeurs répertorient tous les types d'objets que l'on peut manipuler au sein d'une application.

Attribut identifiant,
simple,
élémentaire,
obligatoire,
définitif,
type de valeur : type prédéfini chaîne de caractères,
exemples de valeur : bureau, armoire, tiroir,plan de travail, pile,... ,
document,...
(deux valeurs sont obligatoires : bureau et document)

. nombre_occurrences_permises

Attribut dont la valeur exprime le nombre maximun d'occurrences d'un type que l'on permet au sein de l'application.

Attribut non identifiant,
simple,
élémentaire,
obligatoire,
modifiable,
type de valeur : type prédéfini entier
la valeur négative (-1) sera considérée comme signifiant une valeur infinie.

. nombre_occurrences_présentes

Attribut exprimant le nombre d'occurrences d'un type d'objet,occurrences présentes à un instant considéré au sein de l'application.

Attribut non identifiant,
simple,
élémentaire,
obligatoire,
automatique,
modifiable,
type de valeur : type prédéfini entier.

Lors de l'opération de création d'un objet, l'utilisateur a le choix entre fournir un nom d'objet ou obtenir de façon automatique un code servant de nom d'objet. Pour des raisons d'implémentation, nous ajoutons l'attribut suivant:

. dernier_code

Compteur donnant le nombre d'occurrences d'un type d'objet, occurrences déjà créées au sein de l'application. La valeur de cet attribut sera concaténée avec les trois premières lettres de chaque type d'objet afin d'obtenir un identifiant lors de la création d'une occurence d'un objet de type quelconque.

Attribut non identifiant,
simple,
élémentaire,
obligatoire,
automatique (sa mise à une valeur de départ peut être manuelle),
modifiable,
type de valeur : type prédéfini entier.

III.5. ÉLABORATION D'UN SCHÉMA CONFORME À NDBS

Dans cette section, seront présentées les diverses transformations du schéma E/A étendu; transformations qui permettent d'aboutir à un schéma conforme à NDBS, c'est-à-dire, exploitable via les primitives NDBS.

La démarche suivie est celle préconisée par J-L. HAINAUT [HAINAUT, 1986]. Rappelons brièvement que cette démarche consiste en trois étapes:

- la première est celle de l'analyse conceptuelle [BODART,PIGNEUR,1983]. Son résultat consiste en la production d'un schéma Entité/Association.

- la deuxième est celle de la conception logique qui, à partir de la première, amène à une solution indépendante de tout choix d'un SGBD. Le résultat se présente entre autre sous les formes successives d'un schéma des Accès Possibles exprimé dans le modèle MAG et d'un schéma des Accès Nécessaires, toujours exprimé dans le modèle MAG et où, à partir des algorithmes effectifs (dans notre cas les opérations sur les objets de bureau et sur les objets catégorie objets de bureau) il est procédé à un relevé des accès utilisés par ces algorithmes. Remarquons que, dans cette étape, certaines redondances de structure de données peuvent être admises, ceci afin de diminuer les accès aux articles..

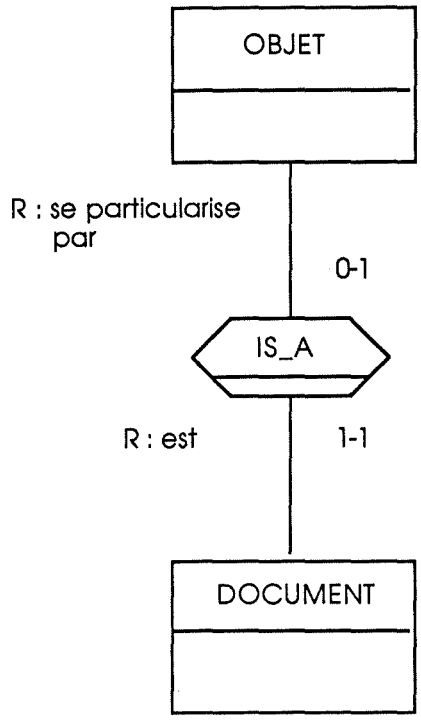
- enfin, la troisième étape représente la conception physique qui permet la traduction du schéma des Accès Nécessaires en un schéma conforme au SGBD choisi pour l'implémentation.

Les diverses transformations se basent sur des règles de transformation qui rendent les divers schémas réversibles et équivalents au point de vue sémantique.

1° étape

Pour rappel, le schéma E/A Etendu et ses contraintes d'intégrité supplémentaires se trouvent pages 14 et 15.

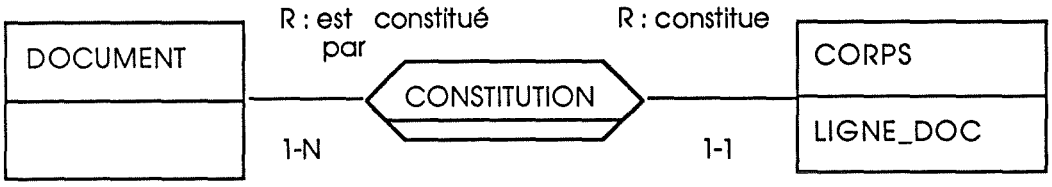
Avant d'appliquer les transformations propres à la conception d'une base de données, il a été procédé à une mise sous forme canonique du schéma E/A Etendu (élimination ou contrôle de la redondance, élimination des ambiguïtés [BODART, PIGNEUR, 1983]). Ensuite, nous avons éliminé la structure sémantique avancée (généralisation/spécification) afin d'obtenir un schéma plus "classique".



Parmi les trois techniques permettant la matérialisation des relations d'inclusion IS_A [HAINAUT,1988] , nous avons retenu celle qui consite à représenter chaque type d'entité et à définir un type d'association entre les types d'Entité concernés

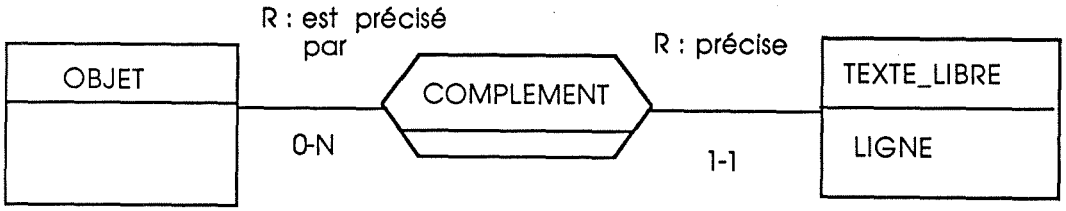
Les deux autres techniques n'ont pas été retenues car elles conduisaient à un schéma E/A "contraire" à notre philosophie , à notre idée de départ (désir de marquer ce qui diffère un document d'un autre objet de bureau, ceci à des fins de simplicité et d'efficacité).

Pour respecter notre décision de considérer tout objet de bureau sous deux formes associées (EN-TETE et CORPS), il est procédé à l'élimination de l'attribut répétif LIGNE_DOC.



Propriété de l'attribut LIGNE_DOC :
- non identifiant,
- simple,
- élémentaire,
- obligatoire,
- modifiable,
- type de valeur type prédéfini chaîne de caractères.

A ce stade il nous a semblé intéressant, pour des raisons de similitudes de comportement, d'éliminer également l'attribut répétitif LIGNE.



Propriété de l'attribut LIGNE :
- non identifiant,
- simple,
- élémentaire,
- obligatoire,
- modifiable,
- type de valeur type prédéfini chaîne de caractères.

2° étape

Cette deuxième étape consiste à établir :
1. Le schéma des Accès possibles (page 29)
2. Le schéma des Accès Nécessaires (page 31)

Graphiquement, un type d'article est représenté par une cartouche contenant le nom de ce type. L'association d'un item (représenté par son nom) à un type d'article est représentée par un arc orienté entre la représentation du type d'article et celle de l'item. On distingue également un item simple identifiant (pas de symbole), un item simple non identifiant (triangle), un item répétitif identifiant (triangle) et un item répétitif non identifiant (diabolo). On représente également le caractère obligatoire d'un item par une barre et le caractère facultatif par une absence de barre. Ces représentations sont également utilisées pour les types de chemin (chemin d'accès inter-articles).

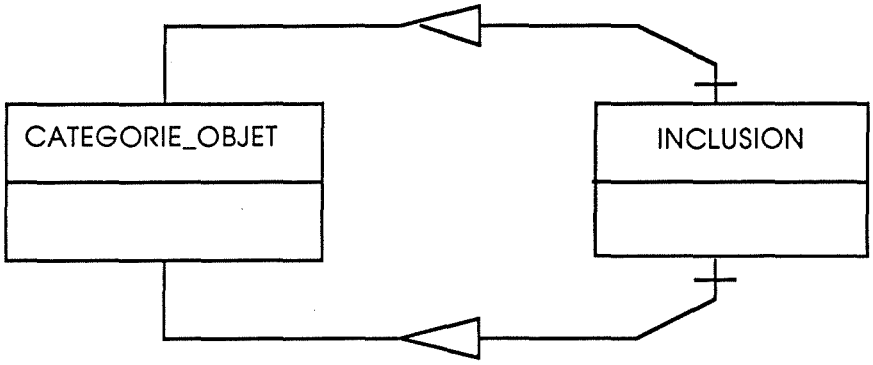
Sur base des opérations présentées, les accès, les clés d'accès et chemins d'accès suivant s (⟶) ont été retenus:

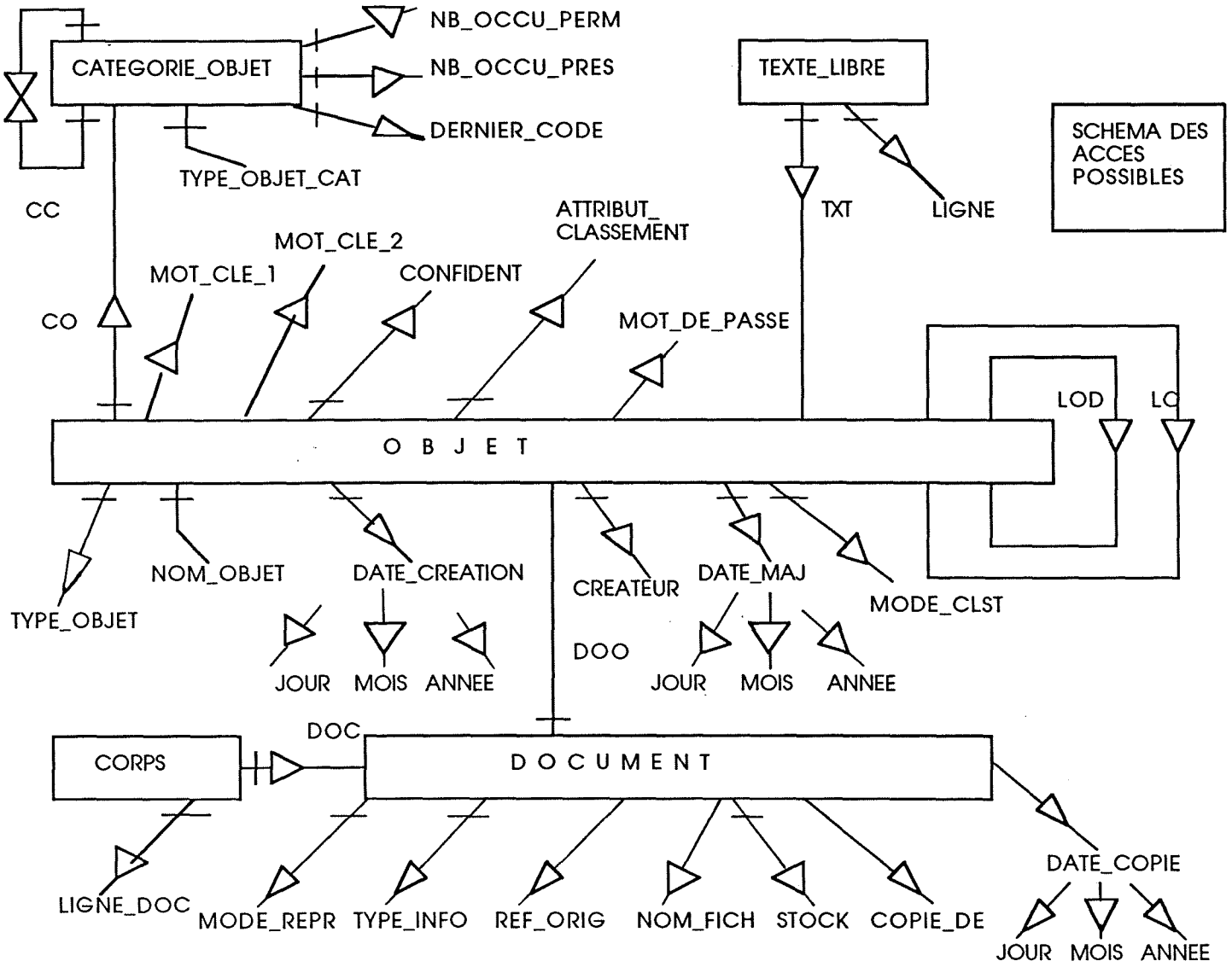
3° étape

Les principales restrictions que l'on doit apporter au schéma des Accès Nécessaires pour obtenir un schéma conforme à NDBS sont les suivantes :
1) pas d'items facultatifs;
2) un seul identifiant par type d'article (et au niveau 1);
3) les contraintes d'existence ne sont pas exprimables;
4) tout chemin est de type 1-1, N-1, 1-N et est doté de son inverse;
5) les chemins récursifs sont admis...
6) ainsi que les attributs décomposables (complexes);
7) et les attributs répétitifs (hormis les identifiants).

Les items facultatifs seront gérés par l'application. Ainsi, pour les attributs de type prédéfini chaîne de caractères, il sera affecté, quant ce sera nécessaire, une valeur blanche (chaîne vide). Pour les attributs de type entier, la valeur affectée sera la valeur entière 0.
Les contraintes d'existence seront exprimées par des contraintes d'intégrité supplémentaires (voir page 33).

Le SGBD NDBS n'admetttant pas de chemin de la classe fonctionnelle N-N, il a été procédé à son élimination [HAINAUT, 1986].
Pour des raisons de facilité, il a été introduit une redondance. Toute entité de type INCLUSION, identifiée par un couple d'entité CATEGORIE_OBJET, le sera également par un identifiant constitué de la concaténation des deux identifiants des entités de type CATEGORIE_OBJET.


CATEGORIE_OBJET
INCLUSION


CATEGORIE_OBJET
NB_OCCU_PERM
NB_OCCU_PRES
DERNIER_CODE
TYPE_OBJET_CAT
CC
TEXTE_LIBRE
TXT
LIGNE
SCHEMA DES ACCES POSSIBLES
ATTRIBUT_ CLASSEMENT
MOT_CLE_2
MOT_CLE_1
CONFIDENT
CO
MOT_DE_PASSE
LOD
LC
O B J E T
TYPE_OBJET
NOM_OBJET
DATE_CREATION
CREATEUR
DATE_MAJ
MODE_CLST
JOUR
MOIS
ANNEE
DOO
JOUR
MOIS
ANNEE
DOC
CORPS
D O C U M E N T
LIGNE_DOC
MODE_REPR
TYPE_INFO
REF_ORIG
NOM_FICH
STOCK
COPIE_DE
DATE_COPIE
JOUR
MOIS
ANNEE

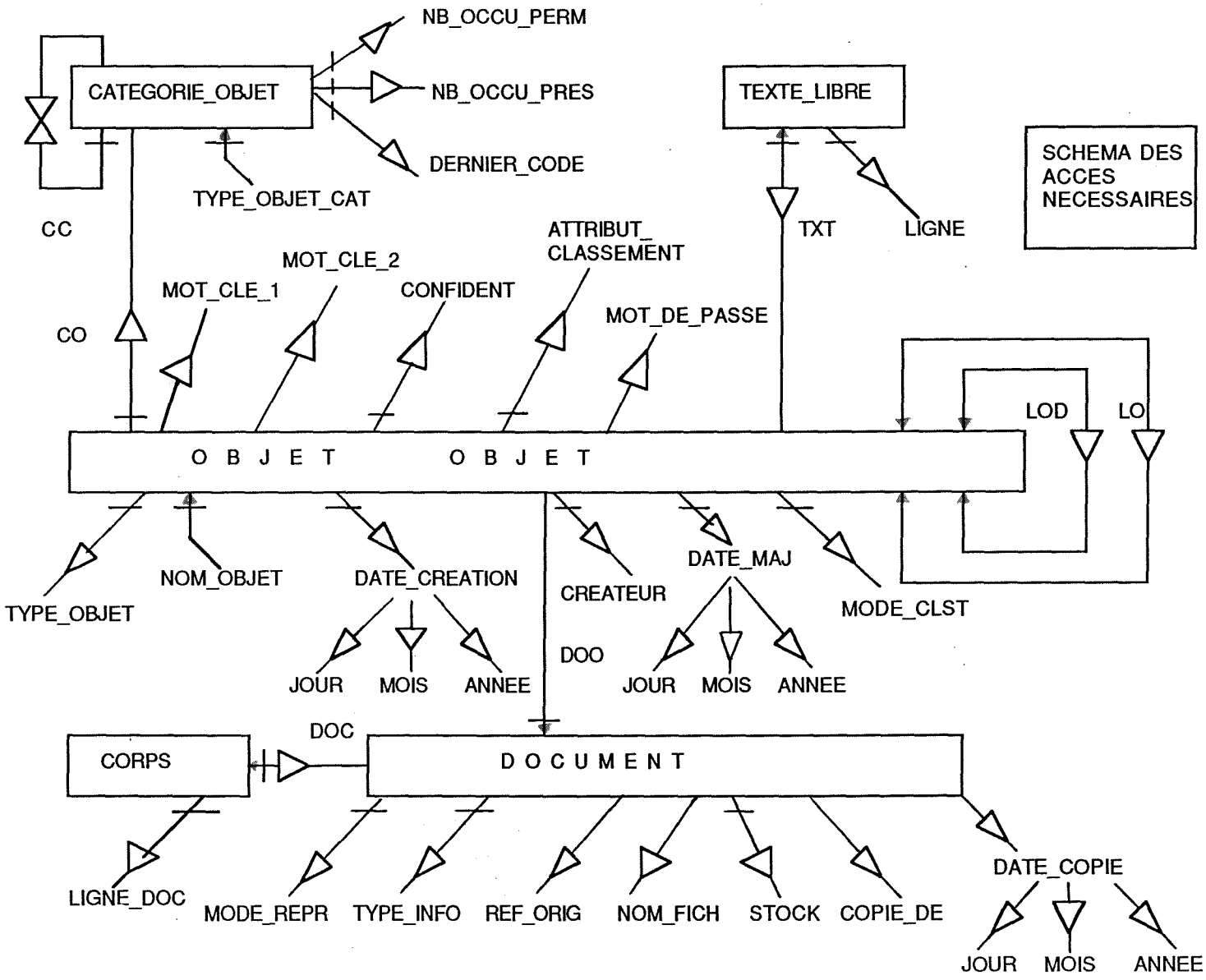
CONTRAINTES D'INTEGRITE SUPPLEMENTAIRES :

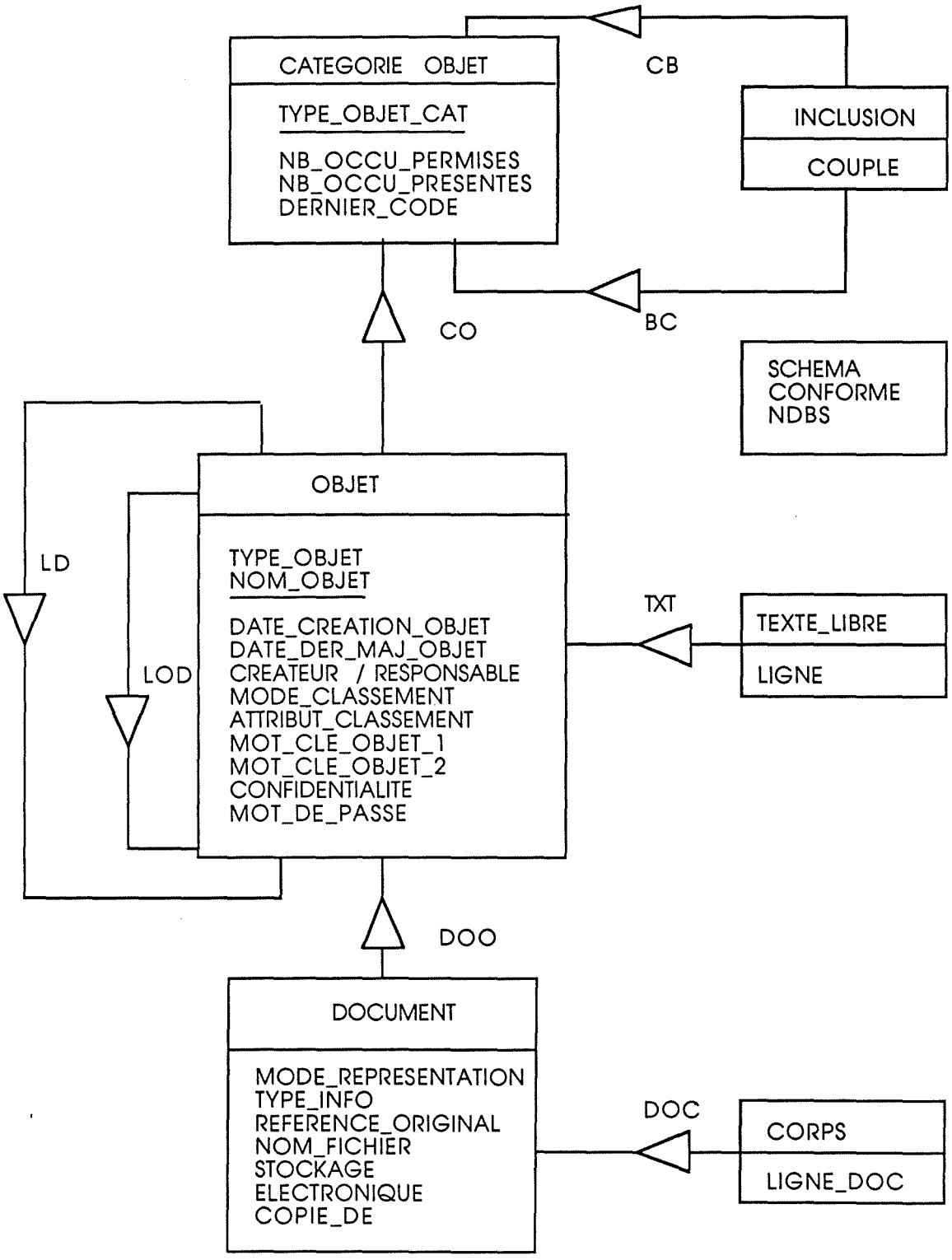
- Contraintes sur items :

pour tout o d'OBJET : DATE_DER_MAJ >= DATE_CREATION_OBJET;
.pour tout o d'OBJET et pour tout d de DOCUMENT (DOO : O) :
DATE_COPIE (:d) >= DATE_CREATION_OBJET (: o)

- Domaines de valeurs (idem aux précédents) ;

- Contraintes référentielles :
pour tout type d'article OBJET ;
TYPE_OBJETc CATEGORIE_OBJET.TYPEOBJETCAT.


NB_OCCU_PERM
CATEGORIE_OBJET
NB_OCCU_PRES
DERNIER_CODE
TYPE_OBJET_CAT
CC
TEXTE_LIBRE
TXT
LIGNE
SCHEMA DES ACCES NECESSAIRES
ATTRIBUT_ CLASSEMENT
MOT_CLE_2
MOT_CLE_1
CONFIDENT
MOT_DE_PASSE
CO
LOD
LO
O B J E T
O B J E T
TYPE_OBJET
NOM_OBJET
DATE_CREATION
CREATEUR
DATE_MAJ
MODE_CLST
JOUR
MOIS
ANNEE
DOO
JOUR
MOIS
ANNEE
DOC
CORPS
D O C U M E N T
LIGNE_DOC
MODE_REPR
TYPE_INFO
REF_ORIG
NOM_FICH
STOCK
COPIE_DE
DATE_COPIE
JOUR
MOIS
ANNEE


CATEGORIE OBJET
TYPE_OBJET_CAT
NB_OCCU_PERMISES
NB_OCCU_PRESENTES
DERNIER_CODE
CB
INCLUSION
COUPLE
BC
CO
SCHEMA
CONFORME
NDBS
OBJET
TYPE_OBJET
NOM_OBJET
DATE_CREATION_OBJET
DATE_DER_MAJ_OBJET
CREATEUR / RESPONSABLE
MODE_CLASSEMENT
ATTRIBUT_CLASSEMENT
MOT_CLE_OBJET_1
MOT_CLE_OBJET_2
CONFIDENTIALITE
MOT_DE_PASSE
LD
LOD
TXT
TEXTE_LIBRE
LIGNE
DOO
DOCUMENT
MODE_REPRESENTATION
TYPE_INFO
REFERENCE_ORIGINAL
NOM_FICHIER
STOCKAGE
ELECTRONIQUE
COPIE_DE
DOC
CORPS
LIGNE_DOC

CONTRAINTES D'INTEGRITE SUPPLEMENTAIRES :

- le type de chemin DO est de classe fonctionnelle 1-1; ;
- contraintes d'existence :

  - toute occurrence d'entité CORPS doit être associée à une occurrence d'entité DOCUMENT via le type de chemin DOC;

  - toute occurrence d'entité DOCUMENT doit être associée à une occurrence d'entité OBJET via le type de chemin DOO;
  - toute occurrence d'entité OBJET (sauf celle où TYPE_OBJET = bureau) doit être associée à une occurrence d'entité OBJET via le type de chemin LO et LOD;

  - toute occurrence d'entité TEXTE_LIBRE doit être associée à une occurrence d'entité OBJET via le type de chemin TXT;

  - toute occurrence d'entité OBJET doit être associée à une occurrence d'entité CATEGORIE_OBJET via le type de chemin CO;
  - toute occcurrence d'entité CATEGORIE_OBJET doit être associée à une occurrence d'entité INCLUSION via un type de chemin CB et BC; et inversément.

  - contraintes sur attributs (voir plus haut);

- contrainte référentielle :
  OBJET.TYPE_OBJET c CATEGORIE.TYPE_OBJET_CAT.

Conclusion

Dans ce chapitre, nous avons réalisé la représentation des types abstraits correspondant aux **objets** isolés dans l'analyse fonctionnelle. C'est en conservant ces types abstraits que nous avons pu "passer" de la modélisation utilisée dans l'approche orientée objet à une modélisation basée sur le modèle E/A, modèle sur lequel repose notamment N.D.B.S.. La puissance du modèle E/A, auquel nous avons ajouté des extensions, a permis une traduction assez aisée. Enfin, nous avons produit un schéma de base de données conforme à N.D.B.S. en précisant les contraintes d'intégrité l'accompagnant.

Chapitre IV : SPECIFICATIONS DES PRIMITIVES

La première partie de ce chapitre porte sur les spécifications externes des primitives. Ces spécifications visent à expliquer de façon rigoureuse et précise ce que réalise chacune des primitives; les utilisateurs de celles-ci auront donc en main tous les éléments indispensables pour une utilisation optimale et correcte des procédures de l'environnement de rangement.

La seconde partie du chapitre sera occupée par des commentaires sur la réalisation des primitives.

IV.1. SPECIFICATIONS EXTERNES DES PRIMITIVES

Après avoir énoncé succinctement les primitives disponibles dans l'environnement de rangement, nous passerons en revue les nouveaux types des données et les variables utilisées pour finalement spécifier complètement l'ensemble des primitives.
Notons d'emblée le caractère atomique de ces primitives, chacune d'entre elles correspondant à une transaction.

IV.1.A. RÉSUMÉ DES PRIMITIVES

Les primitives disponibles dans l'environnement de rangement sont divisées en cinq catégories :

1) les primitives d'initialisation de l'environnement de rangement

. creer_ER : créer un environnement de rangement;

. ouvrir_ER : activer un environnement de rangement;

. fermer_ER : désactiver un environnement de rangement;

2) les primitives de construction et de mise à jour de la structure de l'environnement de rangement

. creer_nouveau_type : créer un nouveau type d'objet de bureau;

. supprimer_type : supprimer un type d'objet de bureau existant;

. creer_relation : créer une relation d'inclusion entre deux types d'objet;

. supprimer_relation : supprimer une relation d'inclusion entre deux types d'objet;

. modifier_attributs_type : mettre à jour les attributs d'un type d'objet de bureau;

. donner_premier_type
. donner_type_suivant : donner les attributs d'un type d'objet de bureau

. connaitre_structure : donner la structure d'un environnement de rangement.

3) les primitives d'accès aux objets de bureau

. donner_premier
donner_suivant : fournir l'identifiant interne d'un objet de bureau de type déterminé;

. acces_direct : fournir l'identifiant interne d'un objet de bureau par son identifiant externe;

. acces_premier
acces_suivant : fournir l'identifiant interne d'un composant d'un objet de rangement.

4) les primitives de création et de mise à jour des objets de bureau

. creer : créer et ranger un objet de bureau dans l'environnement de rangement;

. detruire : supprimer un objet de bureau;

. caracteristiques : donner les attributs d'un objet de bureau;

. modification_attributs : mettre à jour les attributs d'un objet de bureau;

. localisation_origine : connaître le lieu de rangement d'origine d'un objet rangeable;

. localisation_reelle : connaître la localisation réelle d'un objet rangeable;

. changer_lieu_origine : modifier le lieu de rangement d'origine d'un objet rangeable;

. deplacer_objet : modifier la localisation réelle d'un objet rangeable;

. reinserer_place_origine : ranger un objet de bureau à sa place d'origine;

. associer_texte_libre : commenter un objet de bureau;

. detruire_texte_libre : détruire le texte libre d'un objet de bureau;

. lire_texte_libre : donner le texte libre d'un objet de bureau;

. copie : copier un document;

. obtenir_corps : fournir le corps d'un document;

. remplacer_corps : mettre à jour le corps d'un document;

. placer_corps_hors_ER
placer_corps_in_ER : modifier le mode de stockage d'un document.

5) les primitives de manipulation de l'identifiant interne

. mettre_a_null : affecter la valeur 'null' à un identifiant interne;

. verif_idint : contrôler la valeur d'un identifiant interne (fonction).

IV.1.B. DÉFINITIONS DES NOUVEAUX TYPES

L'utilisateur des primitives dispose des types de données suivant.

* NOM_DE_ER :

type de données qui correspond à une chaîne de caractères (Type Pascal : string[64]) et dont une valeur peut désigner le nom d'un environnement de rangement (avec éventuellement comme préfixe un chemin d'accès);

* IDENTIFIANT-INTERNE :

type de données dont la valeur désigne un objet dans l'environnement de rangement; un identifiant interne 'null' est une valeur particulière traduisant le fait qu'aucun objet n'est référencé.
Le type IDENTIFIANT-INTERNE est appelé DBKEY.
Les variables de ce type peuvent être structurées dans des tableaux ou des enregistrements; elles sont soit statiques, soit dynamiques et ne pourrons jamais être organisées sous forme d'ensemble ou de fichier. Enfin, elles ne peuvent être ni lues, ni écrites.

Remarque : le type DBKEY correspond au type DBREF utilisé par les procédures de N.D.B.S;

* NOMOBJET :

type de données qui correspond à une chaîne de caractères (Type Pascal : string[30]).
La signification d'une variable de ce type est le nom d'un objet de bureau de type quelconque; elle joue le rôle d'identifiant externe pour tous les objets de bureau. Elle se compose de deux parties : les trois premiers caractères du type auquel l'objet appartient et le nom proprement dit de celui-ci; ces deux parties étant séparées par le caractère "/".
Cette façon d'opérer permet d'avoir deux objets de type différent avec le même nom. La pose du préfixe se fait automatiquement à la création de l'objet de bureau;

* TYPEOBJET :

type de données correspondant à une chaîne de caractères (Type Pascal: string[15]). Une variable de ce type désigne le type d'un objet de bureau;

* OBJET_ATTR :

type de données Pascal qui contient d'une part l'identifiant interne d'un objet de bureau et d'autre part l'ensemble des variables correspondant aux attributs de cet objet.

```
objet_attr = record
                ref_objet : DBKEY;
                   typeobjet : string[15];
                   nomobjet : string[30];
                   datecreationobjet : record
                                          jour : integer;
                                          mois : integer;
                                          annee : integer
                                       end;
                   datedermaj : record
                                   jourmaj : integer;
                                   moismaj : integer;
                                   anneemaj : integer
                                end;
                   createurresponsable : string[20];
                   modeclassementobjet : integer;
                   attributclassementob : integer;
                   motcleobjet1 : string[30];
                   motcleobjet2 : integer;
                   confidentialité : boolean;
                   motdepasse : string[20];
                   moderepresentation : string[20];
                   typeinfo : string[20];
                   referenceoriginal : string[30];
                   nomfichier : string[50];
                   stockage : boolean;
                   copiede : string[20];
                   datecopie : record
                                  jourcopie : integer;
                                  moiscopie : integer;
                                  anneecopie : integer
                               end;
                   electronique : boolean
             end;
```

* NOMFICHIER :

type de données correspondant à une chaîne de caractères (Type Pascal: string[50]). Une variable de ce type identifie le nom d'un fichier externe à l'environnement de rangement. Le nom du fichier peut être éventuellement précédé d'un chemin d'accès.

IV.1.C. PARAMETRES DES PRIMITIVES

Cette section nous donne l'occasion d'énoncer les différents paramètres des primitives dont les types ont été décrits dans la section B. Ces paramètres sont respectivement :

* nom_er :

variable Pascal de type NOM_DE_ER.
ex. : 'OFFICE', '\BRUSSEL\QLEOPOLD\ALBERT',
'D:\WEPION\FRAISES\EXPORT';

* ref_objet, ref_document, ref_contenu et ref_contenant :

variables Pascal de type DBKEY qui ne peuvent être manipulées que par les primitives de l'environnement de rangement : la première identifie un objet de bureau tout à fait général, la deuxième un objet de bureau de type 'DOCUMENT' et les deux dernières identifient respectivement un objet rangeable et un objet de rangement;

* type_objet :

variable Pascal de type TYPEOBJET. Pour des raisons d'implémentation, toutes les valeurs doivent être entrées en majuscule.
ex. : 'BUREAU', 'ARMOIRE', 'DOCUMENT', 'POUBELLE', ...;

* nom_objet :

variable Pascal de type NOMOBJET.
ex. : 'DOC/001', 'POU/001', 'ARM/police', 'TIR/police';

* liste_attr :

variable Pascal de type OBJET_ATTR;

* nom_fichier :

variable Pascal de type NOMFICHIER.
ex. : 'memoire.doc', 'C:\word\JAPON1.DAT', ...;

* indicateur_localisation :

variable Pascal de type entier qui détermine la relation exacte que partage un objet de rangement avec un de ses composants.
Domaine de valeurs :
0 : l'objet de rangement considéré est à la fois la localisation d'origine et réelle de l'objet rangeable;
1 : l'objet de rangement considéré est la localisation réelle de l'objet rangeable;
2 : l'objet de rangement considéré est le lieu de rangement d'origine de l'objet rangeable;

* code_retour :

variable Pascal de type entier qui indique la façon dont l'exécution de la primitive s'est déroulée.
Les différentes valeurs de "code_retour" sont :
0 : l'exécution de la primitive s'est déroulée correctement;

1 : l'environnement de rangement ou l'objet demandé n'a pas été trouvé;
2 : l'identifiant externe fourni existe déjà dans l'E.R. actif;
3 : l'objet identifié par "ref_objet", "ref_document" ou "ref_contenu" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
5 : les arguments d'entrée ne sont pas compatibles;

30 : valeur d'identifiant interne ("ref_objet", "ref_document" ou "ref_contenu") incorrecte;
31 : valeur d'identifiant interne ("ref_contenant") incorrecte;

40 : primitive impossible;
41 : type d'objet inconnu de la structure de l'E.R. actif;
42 : inclusion interdite par la structure de l'E.R. actif;
43 : dépassement du nombre maximum d'occurrences permises de ce type d'objet par la structure de l'E.R. actif;
44 : le relation d'inclusion considérée existe déjà;
45 : le relation d'inclusion considérée n'existe pas;

50 : le fichier identifié par "nom_fichier" n'a pas été trouvé;
51 : le fichier identifié par "nom_fichier" existe déjà;

60 : l'objet de bureau considéré possède déjà un texte libre;
61 : aucun texte libre n'est attaché à l'objet de bureau considéré;

80 : E.R. actif complet, plus de place;

90 : E.R. actif endommagé;
99 : incident technique au cours de l'opération.

Toute valeur de "code_retour" supérieure à 1 signifie l'abandon de l'opération. Dès lors, aucune modification de l'environnement de rangement actif n'a lieu.

IV.1.D. SYNTAXE D'UNE PRIMITIVE

Comme pour les spécifications fonctionnelles, les paramètres d'entrée de la primitive sont repris en caractère normal, les paramètres résultat sont en gras et les paramètres à la fois d'entrée et résultat sont repris en italique.

nom_primitive (paramètre_entrée_1, paramètre_entrée_2, ...
paramètre_E/S_1, paramètre_E/S_2, ...
paramètre_résultat_1,
paramètre_résultat_2,...)

acces_suivant (ref_contenant,
ref_contenu,
indicateur_localisation, code_retour)

IV.1.E. LISTE DES PRIMITIVES

Avant d'entamer la spécification des primitives, nous proposons au lecteur quelques précisions sur la forme du texte des spécifications.

1). Structure des spécifications

Toutes les spécifications externes comportent sept points.

Le premier (Syntaxe) décrit la syntaxe de la primitive
Le deuxième (Objectif) définit en quelques mots la fonction de la primitive.

Le troisième (Arguments) précise les arguments de la primitive.
Le quatrième (Contraintes) énonce les propriétés des arguments qui doivent être satisfaites dans l'état initial de la primitive afin que celle-ci s'exécute correctement c'est-à-dire atteigne son objectif. Il correspond aux hypothèses faites sur les arguments.

Le cinquième (Résultats) précise les paramètres résultat de la primitive.
Le sixième (Propriétés des résultats) énonce les propriétés des résultats qui sont satisfaites dans l'état final de la primitive si celle-ci s'est exécutée correctement. Quand cela est nécessaire, nous décrivons l'état des résultats en cas d'échec de la primitive.

Le septième (Effet sur l'environnement de rangement) décrit les changements qui ont eu lieu dans l'environnement de rangement si l'exécution de la primitive s'est faite correctement.

2) Spécifications des primitives

Dans un souci de clarté, nous ne reprenons ici que la spécification d'une seule primitive; elle servira d'exemple.
Nous renvoyons le lecteur à l'Annexe I pour prendre connaissance des spécifications de toutes les primitives.

Recherche de la localisation d'un objet de bureau au sein de l'environnement de rangement :

a) recherche du lieu de rangement d'origine :

Syntaxe :

localisation_origine (ref_contenu,
ref_contenant, code_retour)

Objectif : déterminer le lieu de rangement d'origine d'un objet rangeable.

Argument :
ref_contenu : l'identifiant interne d'un objet de bureau.

Contraintes :
"ref_contenu" identifie un objet rangeable appartenant à l'environnement de rangement actif. Cet objet ne peut être de type 'BUREAU'.

Résultats :
ref_contenant : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la localisation d'origine a été trouvée;
1 : l'objet de bureau demandé n'a pas été trouvé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : E.R. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_contenant" identifie un objet de rangement qui est le lieu de rangement d'origine du composant considéré.
Pour toute valeur de "code_retour" supérieure à '0', "ref-contenant" prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucun effet sur l'environnement de rangement actif.

b) recherche de la localisation réelle :

Syntaxe :

localisation_reelle (ref_contenu,
ref_contenant, code_retour)

Objectif : déterminer la localisation réelle d'un objet rangeable.

Argument :
ref_contenu : l'identifiant interne d'un objet de bureau.

Contraintes :
"ref_contenu" identifie un objet rangeable appartenant à l'environnement de rangement actif.

Résultats :
ref_contenant : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la localisation réelle a été trouvée;
1 : l'objet de bureau demandé n'a pas été trouvé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : E.R. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_contenant" identifie un objet de rangement qui appartient à l'environnement de rangement et qui est la localisation réelle du composant identifié par "ref_contenu".
Pour toute valeur de "code_retour" supérieure à '0', "ref-contenant" prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucune modification de l'environnement de rangement actif suite à l'application de cette primitive.

IV.2. LA RÉALISATION DES PRIMITIVES

Le rapport de la base de données construite (où figurent les choix des paramètres physiques de NDBS), les types et variables générés par le gestionnaire de schéma ainsi que le texte du code-source des primitives se trouve dans l'annexe II.

Nous pouvons apporter, en plus des commentaires figurant dans ce code-source, quelques explications sur la réalisation des primitives.

Certaines d'entres elles sont très simple dans leur conception . Ces primitives ne requierent que l'emploi des primitives NDBS. En voici deux exemples à titre illustratif :

a). primitive "manipulant" un objet (commentaires page suivante):
CONTENANT et CONTENU sont deux variables entité.

```
procedure localisation_origine (ref_contenu : DBKEY;
                          var ref_contenant : DBKEY;
                          var code_retour : integer);
label exit;

var CONTENANT, CONTENU : TOBJET;

begin
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 30 then goto exit;
     if not dbfound then
                        begin                              (1)
                              code_retour := 3;
                              goto exit;
                        end;

     if CONTENU.TYPEOBJET = 'BUREAU' then
                         begin
                              code_retour := 40;           (2)
                              goto exit;
                         end;

  (a) dbfpath (CONTENANT, CONTENU , -LOD);
     if dbstatus >= 30 then goto exit;
     if not dbfound then                                   (3)
                        begin
                              code_retour := 1;
                              goto exit;
                        end;
```

```
(b) dbcopyref (OBJET, CONTENANT.xxref, ref_contenant);

    exit : ;
    if dbstatus > 10 then                                    ↑
                          begin
                              code_retour := dbstatus;
                              dbclear (CONTENANT);
                              dbcopyref                      (4)
                  (OBJET,CONTENANT.xxref,ref_contenant);
                          end
                     else code_retour := 0;
end;                                                         ↓
```

Commentaires :

- on remarquera que l'opération en elle-même consiste à 'objet (CONTENU) dont on veut connaître la situation (a). Ensuite, la référence du CONTENANT trouvé est affectée à la variable ref_contenant (b). Le tout s'est donc effectué en 2 instructions .

- les contrôles sont multiples. Ici, on vérifie, dans l'ordre, que:
  (1) la valeur de ref_contenu est correcte et que l'objet dont on demande la localisation existe bien ;
  (2) que cet objet n'est pas le bureau car celui-ci est le seul qui n'est pas contenu dans un autre objet ;
  (3) que l'objet a bien un contenant (en toute logique on est sûr qu'il possède un contenant) ;
  (4) les exécutions des primitives NDBS se sont correctementdéroulées. Dans le cas contraire, la valeur de ref_contenant est mise à 'null' et la valeur du code-retour NDBS (variable dbstatus) est affectée à notre variable code-retour. **La base de nos code-retour est donc constituée des valeurs de retour NDBS.**

b. primitive de structuration de l'ER
(commentaires page suivante)

```
procedure creer_relation  (typeobjet_rg,
                           typeobjet_rb : TYPEOBJET;
                           var code_retour :integer);
label exit;

var CATRG (RG pour de rangement),
    CATRB (RB pour rangeable) : TCATEGORIEOBJET;
    INCL : TINCLUSION;

  begin
        CATRG.TYPEOBJETCAT := typeobjet_rg;
        dbid (CATEGORIEOBJET,CATRG);
        if dbstatus >= 80 then goto exit;
        if not dbfound then
                          begin                          (1)
                                code_retour := 4;
                                goto exit ;
                           end;

        CATRB.TYPEOBJETCAT := typeobjet_rb;
        dbid (CATEGORIEOBJET, CATRB);
        if dbstatus >= 80 then goto exit;
        if not dbfound then                              (2)
                          begin
                                code_retour := 3;
                                goto exit ;
                           end;
        if  (typeobjet_rg = 'DOCUMENT') or
            (typeobjet_rb = 'BUREAU')
          then
              begin                                      (3)
                   code_retour := 40;
                   goto exit ;
               end;
```

```
(a) INCLUSION.COUPLE := concat
                  (typeobjet_rg,typeobjet_rb);
   dbid (INCLUSION, INCL);
   if dbstatus >= 80 then goto exit;
   if  dbfound then
                  begin
                       code_retour := 45;
                       goto exit ;
                  end
                     else
                        begin
                     (b) dbcreate (INCLUSION,INCL);
                         if dbstatus > 0 then goto exit;
                     (c) dbinsert ( INCL, CATRG, CB);
                         if dbstatus >= 80 then goto exit;
                     (d) dbinsert ( INCL, CATRB ,BC);
                         if dbstatus >= 80 then goto exit;
                        end;

    exit : ;

    if dbstatus > 10 then code_retour := dbstatus
                     else code_retour := 0;
end;
```

(4)

Commentaires :

- l'opération consiste principalement à concaténer la valeur de 'type_objet_rg' (le type de l'objet de rangement) avec la valeur de 'type_objet_rb' (a).
Ensuite, on accède à une entité INCLUSION sur base d'un identifiant qui est la valeur résultat de la concaténation. Si une entité est trouvée, cela signifie que la relation existe déjà. Dans le cas contraire, une entité est crée (b), cette entité ayant pour identifiant la valeur résultat de la concaténation. Enfin, on relie l'entité crée aux entités représentant les types des objets (c) (d)

- les contrôles ont pour but de vérifier que :
(1) le type de l'objet considéré comme objet de rangement existe bien (et qu'il n'est pas du type 'DOCUMENT') (3);
(2) le type de l'objet considéré comme objet rangeable existe bien (et qu'il n'est pas du type 'BUREAU') (3);
(4) les primitives NDBS se sont correctement exécutées.

D'autres primitives ont demandé la réalisation préalable d'outils (procédures). Cette obligation provient du fait que lorsque l'on veut relier une entité (cible) à une entité origine, l'entité cible est placée à la suite des autres entités cibles déjà reliées. L'ordre d'insertion d'une entité cible est donc un ordre chronologique.

Ceci représente une contrainte pour nous. En effet, pour pouvoir respecter un ordre de classement défini par l'utilisateur (ordre alphabétique croissant par exemple) on désire insérer une entité cible **parmi** les autres qui respectent déjà l'ordre défini. Cette possibilité nous étant refusée, nous avons construit les outils suivants afin d'apporter une solution à ce problème. :

- le premier outil permet la lecture des entités cibles, la mise de deux valeurs dans une liste chaînée (la référence de l'entité est toujours reprise, ceci pour pouvoir retrouver l'entité, et la valeur de l'attribut base du classement), et la destruction des occurences du chemin reliant les entités cibles à l'entité origine;

- le deuxième outil réalise l'insertion d'un élément sur base de la valeur de l'attribut de classement ; une procédure permet d'insérer par valeur croissante , une autre par valeur décroissante ;

- le troisième outil consiste en une procédure de tri d'une liste chaînée .

Enfin, la reconnection des entités cibles à l'entité origine ne nécessite que quelques lignes et n'a donc pas fait l'objet d'une procédure.

Ces différents outils sont combinés entre eux dans des procédures telles que, par exemple, INSERER_NOUVEAU_ELEMENT dont le but est précisément d'insérer un objet parmi d'autres en respectant le clasement défini.

Illustration de l'emploi de cette procédure Inserer_nouveau_element :

```
procedure deplacer_objet (ref_contenu,ref_contenant : DBKEY;
                                  var code_retour : integer);

label exit;

var ANCIENCONTENANT, CONTENU,
            NOUVEAUCONTENANT : TOBJET;
    INCL : TINCLUSION;
    lieu : char;

begin
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 30 then goto exit;
     if not dbfound then
                        begin
                             code_retour := 3;
                             goto exit;
                        end;

     if CONTENU.TYPEOBJET = 'BUREAU' then
               begin
                    code_retour := 40;
                    goto exit;
               end;
```

```
        dbdirect (OBJET, NOUVEAUCONTENANT, ref_contenant);
        if dbstatus = 30 then
                              begin
                                    dbstatus := 31;
                                    goto exit;
                              end;
        if dbstatus > 30 then goto exit;
        if not dbfound then
                            begin
                                 code_retour := 4;
                                 goto exit;
                            end;

        if NOUVEAUCONTENANT.TYPEOBJET = 'DOCUMENT' then
                    begin
                          code_retour := 40;
                          goto exit;
                    end;


        { vérification que les références ne désignent}
        {pas le même objet}
        if dbequal (ref_contenant, ref_contenu)
                     then
                         begin
                               code_retour := 40;
                               goto exit;


         {verification que l'inclusion est permise : }

         INCL.COUPLE := concat
             (NOUVEAUCONTENANT.TYPEOBJET, CONTENU.TYPEOBJET);
         dbid (INCLUSION, INCL);
         if dbstatus >= 80 then goto exit;
         if not dbfound then
                              begin
                                    code_retour := 42;
                                    goto exit ;
                              end;

         { retirer le contenu de son ancien contenant }
         dbfpath (ANCIENCONTENANT, CONTENU, - L0);
         dbremove (CONTENU, ANCIENCONTENANT, L0);
         if dbstatus > 30 then goto exit;

         {mise à jour de la date de der_maj de l'ancien contenant}
         maj_date_maj_contenant (ANCIENCONTENANT.xxref, code_retour);
        if code_retour > 0 then goto exit;


         { inserer le contenu dans le nouveau contenant
         {en respectant le mode de classement de ce}
         { nouveau contenant }

         lieu := 'r';   {''r'' déigne le lieu rérel de l''objet}
         inserer_nouveau_element
            (ref_contenu, lieu, ref_contenant, code_retour);
         if code_retour > 0 then goto exit ;

         exit : ;
         if dbstatus > 10 then code_retour := dbstatus
                          else code_retour := 0;
end;
```

Chapitre V : EXEMPLES D'APPLICATION

Ce chapitre poursuit un triple objectif : d'une part, nous présentons un schéma de développement d'une application ou, en d'autres termes, nous proposons une marche à suivre en vue de résoudre des problèmes d'ordre bureautique; d'autre part, nous essayons d'illustrer les qualités des primitives développées et enfin, nous illustrons par des exemple l'utilisation des primitives. Le code source des exemples d'application et du programme de démonstration figure dans l'annexe II.

Le type de problème en face duquel nous nous trouvons est simple : il s'agit d'organiser, de ranger et de structurer des informations à l'intérieur d'un environnement de rangement.
Ce problème se décompose à son tour en deux sous-problèmes : le premier touche à la question de savoir **où stocker les informations** tandis que le second porte sur les **opérations** qui seront **disponibles** dans l'environnement de rangement.

Tout en résolvant les deux sous-problèmes cités supra, nous tenterons plus particulièrement d'illustrer certaines qualités des primitives telles que :

. leur généralité et leur souplesse d'utilisation;
. leur faculté d'adaptation à tous les problèmes qui peuvent se présenter;
. la possibilité de les combiner en vue d'obtenir des services de plus haut niveau; c'est-à-dire la construction de macro-primitives.

Nous allons raïsonner, tout au long de ce chapitre, sur un petit exemple d'application. Celui-ci, malgré son apparence anodine, nous donnne l'occasion d'utiliser pleinement l'outil qui vient d'être développé.

L'environnement en face duquel nous nous trouvons est réduit à sa plus simple expression : il se compose d'un plan de travail, de plusieurs étagères avec des rayons, d'une pile et d'une poubelle. Les utilisateurs qui fréquentent ce bureau désirent manipuler des documents c'est-à-dire principalement les stocker dans le bureau, les ranger dans les rayons des étagères et pouvoir les regrouper dans des fardes.

Comme on peut s'en rendre compte, cet environnement est simple tant au niveau de ses composants qu'au niveau de ses actions. Et maintenant, il ne nous reste plus qu'à décrire ce bureau au moyen des objets et des primitives à notre disposition.

V.1. PREMIER SOUS-PROBLEME :
LA STRUCTURE DE L'ENVIRONNEMENT DE RANGEMENT

"Où stocker l'information" pose le problème de la structure de l'environnement de rangement. De quels objets sera-t-il composé, comment ces objets seront-ils agencés, ...

Les primitives de structuration représentent les moyens mis à la disposition du programmeur d'application afin de répondre à cette question. Elles lui permettent de construire l'environnement de rangement dont la structure se rapproche le plus du problème auquel il se trouve confronté.

Dès lors qu'une structure est spécifique à une application, l'exemple qui suit ne poursuit qu'un but purement illustratif. Il ne doit donc pas être considéré comme une solution universelle.

Outre le **'BUREAU'** et le **'DOCUMENT'** qui sont des types d'objet indispensables vu leur caractère particulier (respectivement maximal et minimal : cfr Chapitre II.2.), nous travaillons avec les types d'objet de bureau suivant :
- le type 'PLAN DE TRAVAIL',
- le type 'ETAGERE',
- le type 'RAYON',
- le type 'PILE',
- le type 'POUBELLE'
- et le type 'FARDE'.

C'est également maintenant, lors de la création des types, que nous pouvons introduire certaines restrictions les concernant.
Ainsi, il se peut que les utilisateurs du bureau ne désirent qu'un seul plan de travail et qu'un seule poubelle, pas plus de deux piles et au maximum trois étagères.
Grâce à l'attribut "nb_occu_permises", nous pouvons tenir compte de ces contraintes.

Cela donne par exemple :

```
. . . . . .
  Var liste : NOUVEAUTYPE;
  Repeat
    Write('Nom du nouveau type d'objet : ');
    Readln(liste.type_objet_cat);
    Write('Nombre maximum d'occurrence : ');
    Readln(liste.nb_occu_permises);
    Write('Numérotation automatique à partir de : ');
    Readln(liste.dernier_code);
    creer_nouveau_type (liste, retour);
    If retour <> 0 Then trt_erreur_creatype(retour);
    Write('Création d'un autre type (O/N) ?');
    Readln(encore);
  Until encore <> 'O'
. . . . . .
```

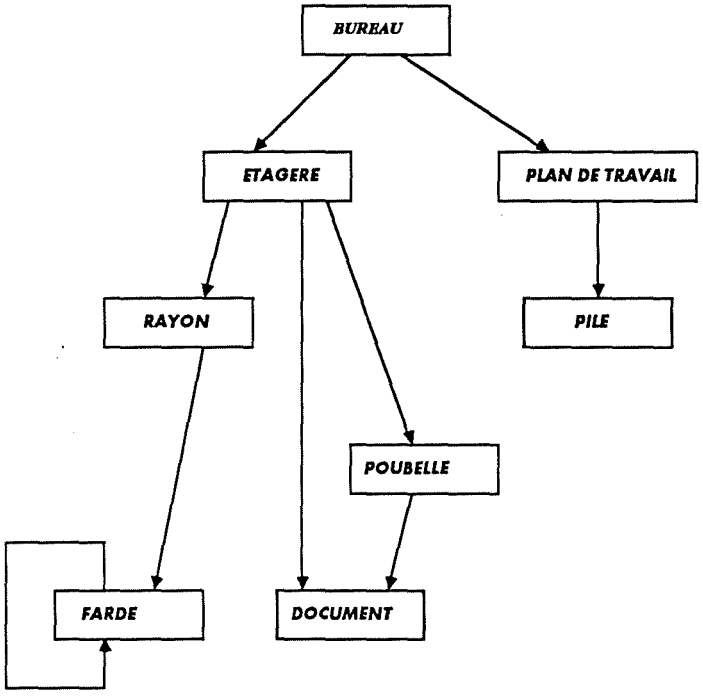
Mais définir les types d'objet propres à une structure ne suffit pas. Il faut encore fixer les relations d'inclusion entre ces différents types.

En plus des inclusions habituelles telles que :
'ETAGERE' et 'PLAN DE TRAVAIL' dans 'BUREAU',
'RAYON' dans 'ETAGERE',
'FARDE' et 'DOCUMENT' sur 'RAYON',
'PILE' et 'DOCUMENT' sur 'PLAN DE TRAVAIL',
'DOCUMENT' dans 'POUBELLE',
nous introduisons des relations spécifiques à l'application qui nous occupe : permettre le rangement d'une poubelle sur un rayon ('POUBELLE' sur 'RAYON'), envisager une farde qui contiendrait elle-même une autre farde ('FARDE' dans 'FARDE').
Si l'on s'en tient à ces relations, jamais l'on ne pourra placer une farde dans une poubelle; de même une pile restera désespérément vide. Il est donc impérieux de définir **toutes** les relations d'inclusion de la structure.

Pratiquement, la définition d'une relation d'inclusion se limite à :

```
. . . . . .
  Var contenant, contenu : TYPEOBJET;

  contenant := 'POUBELLE';
  contenu := 'DOCUMENT';
  creer_relation (contenant, contenu, retour);
  If retour <> 0 Then .....
. . . . . .
```

En prenant les types définis ci-dessus ainsi que les relations ou ils sont impliqués, nous obtenons la structure suivante :



LEGENDE : ⟶ = PEUT CONTENIR

STRUCTURE DE L'ENVIRONNEMENT DE RANGEMENT

Notons que, si les utilisateurs le désirent, il est possible de modifier la structure de l'environnement de rangement au cours de l'exploitation de l'application.
Les changements de structure se font sans risque s'ils consistent en l'ajout de nouveaux types d'objet ou de nouvelles relations d'inclusion.
La suppression de type ou de relation d'inclusion peut se faire mais à la condition expresse d'observer les conseils de prudence émis lors des spécifications fonctionnelles.

Cette première section illustre bien le caractère général des primitives de structuration. Elles permettent d'organiser facilement n'importe quel environnement de rangement en laissant toute liberté au programmeur et à son client de fixer la structure du bureau, que ce soit avant ou en cours d'exploitation.

V.2. DEUXIEME SOUS-PROBLEME : LES OPÉRATIONS DISPONIBLES

La deuxième étape s'attache à définir les **opérations** qui seront disponibles dans l'environnement de rangement.

V.2.A. LES PRIMITIVES DE BASE

Parmi les primitives développées, il en est certaines qui sont directement utilisables dans un programme d'application; pensons aux procédures de création d'un objet (**creer**), de destruction (**detruire**), d'accès par identifiant (**acces_direct**) et de modification (**modification_attributs**, **deplacer_objet**, **changer_lieu_origine**, ...).
Le travail du programmeur se limite, ici, à intégrer les primitives dans un programme d'application et à rendre leur utilisation la plus conviviale possible.

V.2.B. LES MACRO-PRIMITIVES

Avec cette sous-section, nous entrons de plain-pied dans le domaine des macro-primitives d'un environnement de rangement.
Nous nous attacherons plus particulièrement à la spécification de ces dernières et à l'écriture de leur algorithme dans un pseudo-langage; ceci nous donnera l'occasion, une fois encore, d'illustrer la généralité de nos primitives et leur capacité à résoudre la plupart des problèmes.

Une première catégorie de macro-primitives regroupe des primitives qui doivent être absolument combinées pour offrir des services opérationnels. Il s'agit principalement des primitives d'accès aux objets : donner_premier et donner_suivant, acces_premier et acces_suivant.

En combinant les deux primitives de parcours du contenu d'un objet de rangement (**acces_premier** et **acces_suivant**), nous obtenons la macro-primitive "feuilleter_objet". Elle sera très utile aux utilisateurs de l'environnement de rangement qui pourront ainsi prendre connaissance du contenu de leurs fardes.

Sa spécification se présente comme suit :

Syntaxe :
feuilleter_objet (ref_objet,
debut_liste, status_mp)

Objectif : parcourir séquentiellement le contenu d'un objet de rangement.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet de rangement appartenant à l'environnement de rangement actif.

Résultats :
debut_liste : un pointeur vers une liste chaînée.
status_mp : le témoin d'exécution de la macro-primitive.

Propriétés des résultats :
les valeurs possibles de "status_mp" sont dérivées des codes retour des primitives de base ::

0 : l'opération s'est déroulée correctement;
1 : l'objet identifié par "ref_objet" est vide
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
40 : macro-primitive impossible pour un objet de ce type;
90 : E.R. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Un élément de la liste chaînée ("elt_chaîne") se compose de l'identifiant interne ("elt_chaîne.ref") et de l'indicateur de localisation ("elt_chaîne indic") d'un objet rangeable.
L'ordre des éléments de la liste est le même que celui qui prévaut dans l'objet de rangement parcouru.

Effet sur l'environnement de rangement :
aucune modification de l'environnement de rangement actif.

Remarque :
cette macro-primitive est destinée plus spécialement aux fardes et aux tiroirs mais, vu sa généralité, elle sera à la base d'une multitude d'autres macro-primitives.

Algorithme :

procédure feuilleter_objet (ref_objet, **debut_liste**, **status_mp**)

```
liste-chaînee = vide;
acces_premier (ref_objet, ref-composant, indiloca, code-retour)
si code-retour > 0 alors trt-erreur-feuil (code-retour)
sinon
    construire un chaînon et l'insérer dans la liste
    tant que code-retour = 0 répéter
        acces_suivant (ref_objet, ref-composant, indiloca, code-retour)
        décider entre
            code-retour = 1 : mettre signe fin de chaîne et arrêt
            code-retour = 0 : construire un chaînon et l'insérer dans la liste
        autrement trt_erreur_feuil (code-retour)
```

avec "construire un chaînon et l'insérer dans la liste"

```
elt-chaîne.ref = ref-composant
elt-chaîne.indic =indiloca
insérer elt-chaîne à la fin de la liste
```

La deuxième catégorie de macro-primitives fournit des services qui ne sont pas offerts par les primitives "de base" et qui se révèlent cependant indispensables à la "bonne marche" de l'environnement de rangement.
Citons pêle-mêle : vider un poubelle de tous les documents qu'elle contient, modifier le commentaire d'un objet de bureau, mettre de l'ordre dans un objet de rangement, archiver automatiquement des documents, ...

Parmi ces primitives, nous nous proposons d'en réaliser quelques unes. Ce sont celles qui, croyons nous, apportent le plus de services dans le cas qui retient notre attention. La première consiste à vider un objet de bureau des documents qu'il renferme.

Sa spécification se présente comme suit :

Syntaxe :
vider_objet (ref_objet, portée, **status_mp**)

Objectif : vider un objet de rangement des documents qu'il renferme directement ou indirectement.

Arguments :
ref_objet : l'identifiant interne d'un objet de bureau.
portée : une variable de type entier.

Contraintes :
"ref_objet" identifie un objet de rangement appartenant à l'environnement de rangement.
"portée" sélectionne les documents qui seront détruits
Si "portée" vaut '0', la macro-primitive détruit les documents qui ont à la fois, comme lieu de rangement d'origine et localisation réelle, cet objet de rangement ou un composant de celui-ci.
Si "portée" vaut '1' (resp. '2'), la macro-primitive détruit les documents qui ont comme localisation réelle (resp. lieu de rangement réel) cet objet de rangement ou un composant de celui-ci.
Si "portée" vaut '3', la macro-primitive détruit tous les documents contenus directement ou indirectement dans cet objet de rangement.

Résultat :
status_mp : le témoin d'exécution de la macro-primitive.

Propriétés des résultats :
les valeurs possibles de "status_mp" sont :

0 : l'opération s'est déroulée correctement;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
40 : macro-primitive impossible pour un objet de ce type;
90 : E.R. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
si "portée" = '3', tous les documents contenus dans l'objet de bureau identifié par "ref_objet" sont détruits;
si "portée" = '2', tous les documents contenus à l'origine dans l'objet de bureau identifié par "ref_objet" sont détruits;
si "portée" = '1', tous les documents contenus réellement dans l'objet de bureau identifié par "ref_objet" sont détruits;
si "portée" = '0', tous les documents contenus à l'origine et réellement dans l'objet de bureau identifié par "ref_objet" sont détruits.

Remarque :
la macro-primitive a été généralisée afin qu'elle s'applique à n'importe quel type d'objet de bureau et non plus seulement à une poubelle.
La destruction ne porte ici que sur des documents; il faudrait modifier peu de chose pour que le vidage porte sur tous les composants de l'objet de rangement.

Algorithme :

```
procédure vider_objet (ref_objet, portée,
                       status_mp)

  feuilleter_objet (ref_objet, debut_liste,status)
  si status > 1 alors trt_erreur_vid (status)
  sinon
    si status = 1   alors status_mp = 0
    sinon
      lire premier élément de la liste chaînée
      tant que pas fin de liste répéter
        si l'objet référencé par elt_chaîne.ref est de type
          'DOCUMENT' alors tentative_destruction(elt_chaine)
        sinon vider_objet (elt_chaine.ref, portée, status)
        lire l'élément suivant de la liste chaînée

avec "tentative_destruction (elt_chaine)"
    si (portée = 3) ou (portée = elt_chaine.indic)
        alors detruire (elt_chaine.ref, retour)
    si code-retour > 0 alors trt-erreur_vid (retour)
```

La deuxième macro-primitive présentée consiste à remplacer le commentaire d'un objet de bureau :

Syntaxe :
remplacer_commentaire (ref_objet, nom_fichier, **status_mp**)

Objectif : remplacer le texte libre d'un objet de bureau par un autre.
Arguments :
ref_objet : l'identifiant interne d'un objet de bureau.
nom_fichier : le nom d'un fichier externe.

Contraintes :
"ref_objet" identifie un objet de bureau appartenant à l'environnement de rangement actif. Objet de type quelconque mais auquel est associé un texte libre.
"nom_fichier" désigne un fichier externe connu.

Résultat :
status_mp : le témoin d'exécution de la macro-primitive.

Propriétés des résultats :
les valeurs possibles de "status_mp" sont :

0 : le changement a bien eu lieu;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
50 : le fichier identifié par "nom_fichier" n'a pas été trouvé;
61 : aucun texte libre n'est attaché à cet objet de bureau;
80 : E.R. actif complet, plus de place;
90 : E.R. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
l'ancien texte libre de l'objet de bureau identifié par "ref_objet" est retiré de l'environnement de rangement actif et remplacé par le contenu du fichier désigné par "nom_fichier".

Algorithme :

procédure remplacer_commentaire (ref_objet, nom_fichier, **status_mp**)

```
detruire_texte_libre (ref_objet, status_mp)
si status_mp > 0 alors trt-erreur-modif (status_mp)
 sinon associer_texte_libre (ref_objet, nom-fichier, status_mp)
si status_mp > 0 alors trt-erreur-modif (status_mp)
```

Enfin, la dernière macro-primitive que nous présentons est celle qui permet de mettre de l'ordre dans un objet de rangement. Elle peut très bien s'appliquer au plan de travail qui, en fin de journée, est encombré par toute un ensemble de documents.

Sa spécification est la suivante :

Syntaxe :
mettre_ordre (ref_objet, **status_mp**)

Objectif : déplace tous les composants directs et réels d'un objet de rangement vers leur lieu de rangement d'origine.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet de bureau appartenant à l'environnement de rangement actif.

Résultat :
status_mp : le témoin d'exécution de la macro-primitive.

Propriétés des résultats :
les valeurs possibles de "status_mp" sont :

0 : le rangement s'est bien déroulé;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
40 : macro-primitive impossible pour un objet de ce type;
90 : E.R. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
tous les composants réels de l'objet de bureau identifié par "ref_objet" rejoignent leur place d'origine.

Algorithme :

```
procédure mettre_ordre (ref_objet, status_mp)

    feuilleter_objet (ref_objet, debut-liste,status)
    si status > 1 alors trt_erreur_ordre (status)
    si status = 1 alors status_mp = 0
    sinon
        lire premier élément de la liste chaînée
        tant que pas fin de liste chaînée répéter
            si elt-chaine.indic = 1 alors reinserer (elt-chaine.ref, status)
            si status > 0 alors trt_erreur_ordre (status)
            sinon lire élément suivant de la liste chaînée
```

CONCLUSION

Les quatre macro-primitives développées représentent un petit échantillon des possibilités des primitives de base. Elles peuvent s'intégrer sans problème dans tout programme d'application.

En fait, nous venons de développer les premiers éléments d'un "toolkit" où le programmeur d'application pourra trouver les services dont il a besoin. Mais ne perdons pas de vue que ces macro-primitives sont encore dans une forme brute et qu'elles doivent être, à leur tour, intégrées dans un programme d'application. Le concepteur de ce programme peut offrir un confort d'utilisation maximum en utilisant, par exemple, une interface graphique.

Le petit programme de démonstration que nous avons développé intègre les primitives de base et les macro-primitives dont les spécifications viennent d'être présentées.

Bien que simple, ce programme offre une grande gamme de services et il prouve qu'il est possible de réaliser une petite application en un minimum de temps et sans trop de difficultés.

CONCLUSION

Rappelons que l'objectif de ce mémoire est de contribuer à l'automatisation de la fonction de rangement au sein d'un bureau.

Afin de réaliser cet objectif, nous avons suivi une démarche basée sur l'approche orientée objet. Dans un premier temps, celle-ci consiste, à partir de la description d'un environnement de rangement manuel, à isoler des objets de bureau et des opérations pouvant s'y appliquer. Dans un deuxième temps, elle vise, au travers de l'étude du passage à l'automatisation, non seulement à enrichir les propriétés mais surtout les opérations associées aux objets.

Cette démarche a donc abouti à une modélisation des concepts (des objets) propres à un environnement de rangement ainsi qu'à un inventaire des opérations possibles.

La représentation des types abstraits, relatifs à ces concepts, est réalisée en utilisant le modèle de description des données (modèle Entité/Association) du S.G.B.G. N.D.B.S. Quant aux opérations, nous les concrétisons sous la forme de primitives reposant sur les services offerts par le sytème de gestion de N.D.B.S.

Ces primitives se veulent "de base" car il est pratiquement impossible de réaliser indépendamment toutes les procédures qui rempliraient l'ensemble des fonctions habituellement attendues dans un environnement de rangement.

Notre contribution se résume donc à un schéma de base de données (réalisation des types abstraits) et à un ensemble de primitives assurant des services de rangement. Ces dernières permettent la manipulation de documents au travers de structures complexes telles que des tiroirs, des armoires, ... Elles autorisent également la structuration d'un environnement de rangement en fonction des besoins rencontrés.

L'intérêt des primitives porte sur trois points principaux.

Premièrement, elles proposent des services d'une grande généralité. Certes, cette qualité requiert des efforts supplémentaires de programmation et exclut pratiquement toute exploitation immédiate des services des primitives par l'utilisateur final. Cependant, cette façon d'opérer apporte une voie de solution à une foule de problèmes spécifiques aux environnements de rangement.

Deuxièmement, le travail de conception du programmeur d'application est plus aisé puique les concepts manipulés (objets de bureau) par les primitives se rapprochent de ceux des utilisateurs.

Troisièmement, les primitives assurent une parfaite transparence quant à l'utilisation de la base de données; les utilisateurs n'ont, dès lors, pas à se préoccuper de la gestion de cet outil.

Vu l'indisponibilité de la version définitive de N.D.B.S., nous n'avons pu implémenter une opération, à savoir celle qui exigeait l'ouverture simultanée de plusieurs bases de données (l'opération de transfert d'objets de bureau d'un environnement de rangement à un autre).

Toutes les opérations de création, de mise à jour et de suppression d'objets de bureau se doivent d'être implémentées sous la forme de transactions afin de respecter la cohérence de la base de données en cas d'incident. Cet aspect - important - a été partiellement ignoré puisque nous ne disposons pas des outils nécessaires. Cependant, nous avons prévu un code de retour avertissant l'utilisateur de la nature des incidents éventuellement rencontrés.

La démarche que nous avons adoptée, c'est-à-dire réaliser d'abord une étude de l'existant (l'environnement de rangement manuel) puis examiner les apports de l'automatisation, nous semble valide puisqu'elle nous a permis de n'omettre aucun des objets et des opérations spécifiques à un bureau.

Nous avons du "abandonner" l'approche orientée objet lors de la réalisation des types abstraits (objets). En effet, les outils mis à notre disposition n'autorisent pas directement une implémentation en termes d'objets. Il reste néanmoins que N.D.B.S., de type réseau et s'appuyant sur le modèle Entité/Association, a permis le réalisation de ces types abstraits. En ce sens, nous avons conservé certains acquis de l'approche orientée objet. Au cours de cette réalisation, nous avons constaté, comme de nombreux auteurs, que par rapport à des modèles tels que le Relationnel, CODASYL, ..., le modèle Entité/Association se prête assez bien à la description de données complexes.

Nous envisageons le prolongement de ce mémoire dans deux directions.

La première s'inscrit dans un plan pratique. Elle consisterait en la réalisation de véritables applications en bureautique; ce qui permettrait, entre autres, de vérifier si l'ensemble des services rendus par les primitives est complet. Ces applications nécessiteraient la conception de macro-primitives qui pourraient prendre place dans une boîte à outils à côté de nos primitives.

La deuxième direction est plus théorique. En effet, la représentation séparée de l'en-tête et du corps d'un document permettra à l'avenir de travailler sur le contenu de celui-ci. L'application de la démarche que nous avons suivie, à l'organisation logique et physique du corps d'un document, débouchera sur la modélisation de sa structure et sur la conception de primitives manipulant l'information associée au document. Combinées aux primitives de rangement, elles offriront un service complet portant à la fois sur la manipulation externe et interne des documents. Signalons enfin que les primitives de rangement ne devront en rien être modifiées puisque, pour accomplir leur service, elles ne manipulent que l'en-tête du document.

BIBLIOGRAPHIE

ADIBA,M. (1986) Problématique des bases de données multi-média.
Nouvelles perspectives des bases de données, Eyrolles. Paris, 1986.

DACHOUFFE,N. (1986) Adaptation du langage DSL aux activités de bureau.
Mémoire de licence en sciences informatiques FUNDP.

BODART,F.et PIGNEUR,Y.(1983) Conception assistée des aplications informatiques
1. Etude d'opportunité et analyse conceptuelle, Masson 1983.

BODART, F. (1988) Cours de conception des systèmes d'information.
Institut d'informatique des Facultés universtaires de Namur.

BOGO,G.; RICHY,H. et VATTON,I (1983) Description des éléments du modèle document.
Rapport de recherche TIGRE n° 6, Septembre 1983.

CONKLIN, J.1987 Hypertext : an introduction and survey.
IEEE,1987.

DE BLASIS, J-P. (1982) La bureautique.
Les éditions d'organisations.

DELISLE,N. et SCHWARTZ,M. 1986 Neptune : a Hypertext System for CAD Application.
Proceedings of ACM SISMOD International Conference on Management of Data, MAY 28-30, 1986.

HAINAUT, J-L. (1986) Conception assistée des applications informatiques.
2- Conception de la base de données. Masson, 1986.

HAINAUT, J-L. (1987) A simple data base system for small computer.
Institut d'informatique des Facultés universtaires de Namur.

HAINAUT, J-L. (1988) Cours de conception de base de données.
Institut d'informatique des Facultés universtaires de Namur.

HAINAUT, J-L. (1989) Cours de conception de base de données, matières approfondies.
Institut d'informatique des Facultés universtaires de Namur.

LESUISSE, R. (1989) Cours des systèmes d'information du bureau et d'aide à la décision.
Institut d'informatique des Facultés universtaires de Namur.

Meyer, B (1988) Objet-oriented software construction.
Printice- Hall Englewood Cliffs (N.J),1988.

PREVOT,I et GENDARME,P (1988) Animation graphique de tâches de bureau : implementation Smaltalk_8O d'un outil d'aide à l'automatisation du travail de bureau.
Mémoire de licence en sciences informatiques FUNDP.

ROBERT, S (1986) Analyse et implémentation d'un environnement de rangement.
Mémoire de licence en sciences informatiques FUNDP.

SOULIER, A. (1984) L'informatique et ses développements.
Masson, 1984.

VAN LANSWEERDE, A. (1989) Cours de méthodologie de développement de logiciels.
Institut d'informatique des Facultés universtaires de Namur.

VELEZ,F. (1984) Défintion formelle du langage LAMBDA
Rapport de recherche TIGRE n° 11, Mars 1984

VELEZ, F et al. (1986) La prise en compte de documents structurés dans un SGBD : aspects du modèle , langage et architecture du système.
Nouvelles perspectives des bases de données, Eyrolles. Paris, 1986.

ANNEXE I

PLAN DE L'ANNEXE I

Annexe I : SPECIFICATIONS EXTERNES DES PRIMITIVES

Nous présentons ici les spécifications de toutes les primitives de l'environnement de rangement. Avant de commencer l'énumération proprement dite, nous faisons quelques rappels. Le premier consiste en l'énumération de toutes les primitives, le deuxième présente les nouveaux types de données et le troisième donne les paramètres des primitives

1. RÉSUMÉ DES PRIMITIVES

Les primitives disponibles dans l'environnement de rangement sont divisées en cinq catégories :

1.A. LES PRIMITIVES D'INITIALISATION DE L'ENVIRONNEMENT DE RANGEMENT

. creer_ER : créer un environnement de rangement;
. ouvrir_ER : activer un environnement de rangement;
. fermer_ER : désactiver un environnement de rangement;

1.B. LES PRIMITIVES DE CONSTRUCTION ET DE MISE À JOUR DE LA STRUCTURE DE L'ENVIRONNEMENT DE RANGEMENT

. creer_nouveau_type : créer un nouveau type d'objet de bureau;
. supprimer_type : supprimer un type d'objet de bureau existant;
. creer_relation : créer une relation d'inclusion entre deux types d'objet;
. supprimer_relation : supprimer une relation d'inclusion entre deux types d'objet;
. modifier_attributs_type : mettre à jour les attributs d'un type d'objet de bureau;
. donner_premier_type
. donner_type_suivant : donner les attributs d'un type d'objet de bureau;
. connaitre_structure : donner la structure d'un environnement de rangement.

1.C. LES PRIMITIVES D'ACCES AUX OBJETS DE BUREAU

. donner_premier
donner_suivant : fournir l'identifiant interne d'un objet de bureau de type déterminé;
. acces_direct : fournir l'identifiant interne d'un objet de bureau par son identifiant externe;
. acces_premier
acces_suivant : fournir l'identifiant interne d'un composant d'un objet de rangement.

1.D. LES PRIMITIVES DE CRÉATION ET DE MISE À JOUR DES OBJETS DE BUREAU

. creer : créer et ranger un objet de bureau dans l'environnement de rangement;
. detruire : supprimer un objet de bureau;
. caracteristiques : donner les attributs d'un objet de bureau;
. modification_attributs : mettre à jour les attributs d'un objet de bureau;
. localisation_origine : connaître le lieu de rangement d'origine d'un objet rangeable;
. localisation_reelle : connaître la localisation réelle d'un objet rangeable;
. changer_lieu_origine : modifier le lieu de rangement d'origine d'un objet rangeable;
. deplacer_objet : modifier la localisation réelle d'un objet rangeable;
. reinserer_place_origine : ranger un objet de bureau à sa place d'origine;
. associer_texte_libre : commenter un objet de bureau;
. detruire_texte_libre : détruire le texte libre d'un objet de bureau;
. lire_texte_libre : donner le texte libre d'un objet de bureau;
. copie : copier un document;
. obtenir_corps : fournir le corps d'un document;
. remplacer_corps : mettre à jour le corps d'un document;
. placer_corps_hors_ER
placer_corps_in_ER : modifier le mode de stockage d'un document.

1.E. LES PRIMITIVES DE MANIPULATION DE L'IDENTIFIANT INTERNE

. mettre_a_null : affecter la valeur 'null' à un identifiant interne;
. verif_idint : contrôler la valeur d'un identifiant interne (fonction).

2. DÉFINITIONS DES NOUVEAUX TYPES

L'utilisateur des primitives dispose des types de données suivant.

* NOM_DE_ER :
type de données qui correspond à une chaîne de caractères (Type Pascal : string[64]) et dont une valeur peut désigner le nom d'un ER (avec éventuellement comme préfixe un chemin d'accès);

* IDENTIFIANT-INTERNE :
type de données dont la valeur désigne un objet dans l'environnement de rangement; un identifiant interne 'null' est une valeur particulière traduisant le fait qu'aucun objet n'est référencé. Le type IDENTIFIANT-INTERNE est appelé DBKEY.

Les variables de ce type peuvent être structurées dans des tableaux ou des enregistrements; elles sont soit statiques, soit dynamiques et ne pourrons jamais être organisées sous forme d'ensemble ou de fichier. Enfin, elles ne peuvent être ni lues, ni écrites.
Remarque : le type DBKEY correspond au type DBREF utilisé par les procédures de N.D.B.S;

* TYPEOBJET :
type de données correspondant à une chaîne de caractères (Type Pascal: string[15]). Une variable de ce type désigne le type d'un objet de bureau;

* OBJET_ATTR :
type de données Pascal qui contient d'une part l'identifiant interne d'un objet de bureau et d'autre part l'ensemble des variables correspondant aux attributs de cet objet.

```
objet_attr = record
                ref_objet : DBKEY;
                    typeobjet : string[15];
                    nomobjet : string[30];
                    datecreationobjet : record
                                            jour : integer;
                                            mois : integer;
                                            annee : integer
                                        end;
                    datedermaj : record
                                    jourmaj : integer;
                                    moismaj : integer;
                                    anneemaj : integer
                                end;
                    createurresponsable : string[20];
                    modeclassementobjet : integer;
                    attributclassementob : integer;
                    motcleobjet1 : string[30];
                    motcleobjet2 : integer;
                    confidentialité : boolean;
                    motdepasse : string[20];
                    moderepresentation : string[20];
                    typeinfo : string[20];
                    referenceoriginal : string[30];
                    nomfichier : string[50];
                    stockage : boolean;
                    copiede : string[20];
                    datecopie : record
                                    jourcopie : integer;
                                    moiscopie : integer;
                                    anneecopie : integer
                                end;
                    electronique : boolean
            end;
```

* NOMOBJET :
type de données qui correspond à une chaîne de caractères (Type Pascal : string[30]).
La signification d'une variable de ce type est le nom d'un objet de bureau de type quelconque; elle joue le rôle d'identifiant externe pour tous les objets de bureau. Elle se compose de deux parties : les trois premiers caractères du type auquel l'objet appartient et le nom proprement dit de celui-ci; ces deux parties étant séparées par le caractère "/".
Cette façon d'opérer permet d'avoir deux objets de type différent avec le même nom. La pose du préfixe se fait automatiquement à la création de l'objet de bureau;

* NOMFICHIER :
type de données correspondant à une chaîne de caractères (Type Pascal: string[50]). Une variable de ce type identifie le nom d'un fichier externe à l'environnement de rangement. Le nom du fichier peut être éventuellement précédé d'un chemin d'accès.

3. PARAMETRES DES PRIMITIVES

Cette section nous donne l'occasion d'énoncer les différents paramètres des primitives dont les types ont été décrits dans la section 2. Ces paramètres sont respectivement :

* nom_er :
variable Pascal de type NOM_DE_ER.
ex. : 'OFFICE', '\BRUSSEL\QLEOPOLD\ALBERT',
'D:\WEPION\FRAISES\EXPORT';

* ref_objet, ref_document, ref_contenu et ref_contenant :
variables Pascal de type DBKEY qui ne peuvent être manipulées que par les primitives de l'ER : la première identifie un objet de bureau tout à fait général, la deuxième un objet de bureau de type 'DOCUMENT' et les deux dernières identifient respectivement un objet rangeable et un objet de rangement;

* type_objet :
variable Pascal de type TYPEOBJET. Pour des raisons d'implémentation, toutes les valeurs doivent être entrées en majuscule.
ex. : 'BUREAU', 'ARMOIRE', 'DOCUMENT', 'POUBELLE', ...;

* nom_objet :
variable Pascal de type NOMOBJET.
ex. : 'DOC/001', 'POU/001', 'ARM/police', 'TIR/police';

* liste_attr :
variable Pascal de type OBJET_ATTR;

* nom_fichier :
variable Pascal de type NOMFICHIER.
ex. : 'memoire.doc', 'C:\word\JAPON1.DAT', ...;

* indicateur_localisation :
variable Pascal de type entier qui détermine la relation exacte que partage un objet de rangement avec un de ses composants.
Domaine de valeurs :
0 : l'objet de rangement considéré est à la fois la localisation d'origine et réelle de l'objet rangeable;
1 : l'objet de rangement considéré est la localisation réelle de l'objet rangeable;
2 : l'objet de rangement considéré est le lieu de rangement d'origine de l'objet rangeable;

* code_retour :
variable Pascal de type entier qui indique la façon dont l'exécution de la primitive s'est déroulée.
Les différentes valeurs de "code_retour" sont :
0 : l'exécution de la primitive s'est déroulée correctement;

1 : l'environnement de rangement ou l'objet demandé n'a pas été trouvé;
2 : l'identifiant externe fourni existe déjà dans l'ER. actif;
3 : l'objet identifié par "ref_objet", "ref_document" ou "ref_contenu" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
5 : les arguments d'entrée ne sont pas compatibles;

30 : valeur d'identifiant interne ("ref_objet", "ref_document" ou "ref_contenu") incorrecte;
31 : valeur d'identifiant interne ("ref_contenant") incorrecte;

40 : primitive impossible;
41 : type d'objet inconnu de la structure de l'ER. actif;
42 : inclusion interdite par la structure de l'ER. actif;
43 : dépassement du nombre maximum d'occurrences permises de ce type d'objet par la structure de l'ER. actif;
44 : le relation d'inclusion considérée existe déjà;
45 : le relation d'inclusion considérée n'existe pas;

50 : le fichier identifié par "nom_fichier" n'a pas été trouvé;
51 : le fichier identifié par "nom_fichier" existe déjà;

60 : l'objet de bureau considéré possède déjà un texte libre;
61 : aucun texte libre n'est attaché à l'objet de bureau considéré;

80 : ER. actif complet, plus de place;

90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Toute valeur de "code_retour" supérieure à 1 signifie l'abandon de l'opération. Dès lors, aucune modification de l'environnement de rangement actif n'a lieu.

4. LISTE DES PRIMITIVES

Avant d'entamer la spécification des primitives, nous proposons au lecteur quelques précisions sur la forme du texte des spécifications.

4.A. STRUCTURE DES SPÉCIFICATIONS

Toutes les spécifications externes comportent sept points.

Le premier (Syntaxe), décrit la syntaxe de la primitive

Le deuxième (Objectif) définit en quelques mots la fonction de la primitive.

Le troisième (Arguments), précise les arguments de la primitive.
Le quatrième (Contraintes) énonce les propriétés des arguments qui doivent être satisfaites dans l'état initial de la primitive afin que celle-ci s'exécute correctement c'est-à-dire atteigne son objectif. Il correspond aux hypothèses faites sur les arguments.

Le cinquième (Résultats), précise les paramètres résultat de la primitive.
Le sixième (Propriétés des résultats) énonce les propriétés des résultats qui sont satisfaites dans l'état final de la primitive si celle-ci s'est exécutée correctement. Quand cela est nécessaire, nous décrivons l'état des résultats en cas d'échec de la primitive.

Le septième (Effet sur l'environnement de rangement) décrit les changements qui ont eu lieu dans l'environnement de rangement si l'exécution de la primitive s'est faite correctement.

4.B. SPÉCIFICATIONS DES PRIMITIVES

4.B.1. Initialisation d'un environnement de rangement

a) Créer un environnement de rangement

Syntaxe :
creer_ER (nom_er, **code_retour**)

Objectif : créer un environnement de rangement.

Argument :
nom_er : nom d'un ER.

Contrainte :
"nom_er" n'identifie encore aucun ER.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'ER. est créé;
1 : l'ER. initial n'a pas été trouvé;
2 : un ER. avec ce nom existe déjà;
99 : incident technique lors de l'exécution de la primitive.

Effet :
un nouvel ER est créé et il est identifié par "nom_er". Ce nouveau ER comprend déjà les types 'BUREAU' et 'DOCUMENT'.

Remarque :
cette primitive nécessite la présence du fichier 'ERINIT' sur l'unité courante, faute de quoi "code_retour' prend la valeur '1'.

b) Ouvrir un environnement de rangement

Syntaxe :
ouvrir_ER (nom_er, **code_retour**)

Objectif : ouvrir un environnement de rangement.

Argument :
nom_er : nom d'un ER (avec éventuellement son chemin d'accès).

Contrainte :
"nom_er" identifie un ER existant.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'ER. est activé;
1 : l'ER. demandé n'a pas été trouvé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
l'ER identifié par "nom_er" est passé à l'état actif et est dès lors prêt pour toute manipulation.

c) Fermer l'environnement de rangement

Syntaxe :
fermer_ER (nom_er, **code_retour**)

Objectif : fermer un environnement de rangement actif.

Argument :
nom_er : nom d'un ER.

Contrainte :
"nom_er" identifie un ER actif.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'ER. a été fermé;
1 : l'ER. demandé n'a pas été trouvé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
ferme l'ER identifié par "nom_er". Une fois fait, plus aucune manipulation n'est possible avant la réouverture de l'ER considéré.

Remarque :
comme il n'est pas possible actuellement d'avoir simultanément plusieurs environnements de rangement actifs, cette primitive a pour effet de fermer le bureau actif quelque soit la valeur de "nom_er".

4.B.2. Primitives de structuration d'un environnement de rangement

a) Avertissement

Nous vous rappelons les conseils de prudence qui accompagnent l'utilisation des primitives de structuration :

- tout type d'objet doit être inclus directement ou indirectement dans le type 'BUREAU';
- la structure d'un environnement de rangement peut être modifiée en cours d'exploitation mais en respectant les deux règles suivantes :
  . un type d'objet peut être détruit si plus aucune de ses occurrences n'est présente dans l'environnement de rangement actif;
  . une relation d'inclusion peut être détruite si plus aucun objet de bureau ne la partage avec un autre.

b) Types et Arguments spécifiques à ces opérations

* NOUVEAUTYPE :
type de données Pascal qui contient l'ensemble des attributs d'un type d'objet de bureau.

```
nouveau_type = record
                 type_objet : string[15];
                 nb_occu_permises : integer;
                 nb_occu_presentes : integer;
                 dernier_code : integer
               end;
```

* type_objet_cat, type_objet_contenu et type_objet_contenant :
variables Pascal de type TYPEOBJET.
ex. : 'ETAGERE','RAYON','BAC','DOSSIER', ...

* liste_cat :
variable Pascal de type NOUVEAUTYPE.

c) Créer un nouveau type d'objet au sein de l'environnement de rangement

Syntaxe :
creer_nouveau_type (liste_cat, **code_retour**)

Objectif : créer un nouveau type d'objet de bureau dans la structure de l'environnement de rangement actif.

Argument :
liste_cat : l'ensemble des valeurs des attributs d'un type d'objet de bureau.

Contrainte :
"type_objet" présent dans "liste_cat" n'identifie encore aucun type d'objet dans la structure l'ER actif.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la copie s'est déroulée correctement;
2 : l'identifiant externe fourni existe déjà dans l'ER. actif;
80 : ER. complet, plus de place;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
la structure de l'ER actif comporte un nouveau type d'objet identifié par "type_objet" présent dans "liste_cat".

d) Supprimer d'un type d'objet présent dans l'environnement de rangement

Syntaxe :
supprimer_type_objet (type_objet_cat, **code_retour**)

Objectif : supprime un type d'objet de bureau de la structure de l'environnement de rangement actif.

Argument :
type_objet_cat : le nom d'un type d'objet de bureau.

Contraintes :
"type_objet_cat" identifie un type d'objet appartenant à la structure de l'ER actif. Il ne peut prendre les valeurs 'BUREAU' et 'DOCUMENT'.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : le type d'objet a été retiré de la structure de l'ER. actif;
40 : primitive impossible pour ce type d'objet;
41 : type d'objet inconnu de la structure de l'ER. actif;
90 : ER. endommagé;
99 :incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
le type d'objet identifié par "type_objet_cat" est supprimé.
Les relations d'inclusion où est impliqué le type supprimé sont détruites à leur tour (aussi bien celles où il est objet de rangement que celles où il est objet rangeable).

e) Créer une relation d'inclusion entre deux types d'objet

Syntaxe :
creer_relation (type_objet_contenant, type_objet_contenu,
code_retour)

Objectif : créer une relation d'inclusion (de contenant vers contenu) entre deux types d'objet de bureau.

Arguments :
type_objet_contenant et type_objet_contenu : des noms de type d'objet de bureau.

Contraintes :
"type_objet_contenant" et "type_objet_contenu" identifient des types d'objet connus de la structure de l'ER. actif.
Les deux arguments ne peuvent prendre les valeurs suivantes : 'DOCUMENT' pour le premier et 'BUREAU' pour le second.

Résultats :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la relation a été créée;
3 : l'objet identifié par "type_objet_contenu" n'a pas été trouvé;
4 : l'objet identifié par "type_objet_contenant" n'a pas été trouvé;
40 : primitive impossible;
44 : la relation d'inclusion existe déjà;
80 : ER. actif complet, plus de place;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
une nouvelle relation d'inclusion existe au sein de la structure de l'ER actif, elle est établie entre les types d'objet identifiés respectivement par "type_objet_contenant" et "type_objet_contenu".

f) Supprimer une relation d'inclusion entre deux types d'objet

Syntaxe :
supprimer_relation (type_objet_contenant, type_objet_contenu, **code_retour**)

Objectif : supprime la relation d'inclusion (de contenant vers contenu) qui existe entre deux types d'objet.

Arguments :
type_objet_contenant et type_objet_contenu : des noms de type d'objet de bureau.

Contraintes :
"type_objet_contenant" et "type_objet_contenu" identifient des types d'objet connus de la structure de l'ER actif. Il existe une relation d'inclusion entre ces deux types d'objet.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la relation a été supprimée;
3 : l'objet identifié par "type_objet_contenu" n'a pas été trouvé;
4 : l'objet identifié par "type_objet_contenant" n'a pas été trouvé;
45 : il n'existe aucune relation d'inclusion entre ces deux types d'objet;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
la relation d'inclusion qui unissait précédemment les types d'objets identifiés par "type_objet_contenant" et "type_objet_contenu" n'existe plus.

g) Modifier un ou plusieurs attributs d'un type d'objet

Syntaxe :
modifier_attributs_type (type_objet_cat, liste_cat, **code_retour**)

Objectif : mettre à jour les attributs d'un type d'objet de bureau à partir de "liste_cat".

Arguments :
type_objet_cat : le nom d'un type d'objet de bureau.
liste_cat : l'ensemble des valeurs d'attribut d'un type d'objet de bureau.

Contraintes :
"type_objet_cat" identifie un type d'objet de bureau appartenant à l'ER actif. Il ne peut prendre la valeur 'BUREAU'.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : le type d'objet a été modifié;
41 : type d'objet inconnu de la structure de l'ER. actif;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.
Si les modifications portent sur des attributs non accessibles (cfr remarque) par cette primitive, les anciennes valeurs sont conservées.

Effet sur l'environnement de rangement :
les attributs du type d'objet identifié par "type_objet_cat" sont mis à jour grâce à "liste_cat".

Remarque :
cette primitive autorise la modification des attributs suivant : "nb_occu_permises", "nb_occu_presentes" et "dernier_code".

h) Donner les caractéristiques du premier type d'objet présent dans la structure de l'environnement de rangement

Syntaxe :
donner_premier_type (**liste_cat**, **code_retour**)

Objectif : fournir l'ensemble des attributs du premier type d'objet connu de la structure de l'ER actif.

Résultats :
liste_cat : l'ensemble des valeurs des attributs d'un type d'objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : le premier type d'objet a été trouvé;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.
"liste_cat" correspond à l'ensemble des attributs du premier type d'objet présent dans la structure de l'ER actif.

Effet sur l'environnement de rangement :
aucun effet sur la structure de l'ER.

i) Donner les caractéristiques du type d'objet suivant présent dans la structure de l'environnement de rangement

Syntaxe :
donner_type_suivant (*liste_cat*, **code_retour**)

Objectif : fournir l'ensemble des attributs du type d'objet de bureau qui suit le type identifié par "type_objet" présent dans "liste_cat".

Argument :
liste_cat : l'ensemble des valeurs des attributs d'un type d'objet de bureau.

Contrainte :
"type-objet" présent dans "liste_cat" identifie un type d'objet connu de la structure de l'ER actif.

Résultats :
liste_cat : l'ensemble des valeurs des attributs d'un type d'objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : le premier type d'objet a été trouvé;
1 : le type d'objet identifié par "type_objet" était le dernier de la structure de l'ER. actif;
41 : type d'objet inconnu de la structure de l'ER. actif;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
aucun effet sur la structure de l'ER.

j) Connaître la structure d'un environnement de rangement

Syntaxe :
connaitre_structure (nom_ER, fichier_structure,
code_retour)

Objectif : fournir la structure d'un ER sous la forme d'un fichier texte ("fichier_structure").

Arguments :
nom_ER : le nom d'un ER.
fichier_structure : le nom d'un fichier externe.

Contraintes :
"nom_ER" identifie l'environnement de rangement actif.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
Les valeurs possibles de "code_retour" sont :

0 : tout s'est bien passé;
1 : l'ER identifié par "nom_ER" n'est pas actif.
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération

Le fichier texte ("fichier_structure") est disponible sur l'unité courante; il reprend, pour chaque type d'objet répertorié dans la structure de l'ER actif, ses attributs et ses relations d'inclusion où l'objet joue le rôle d'objet de rangement et d'objet rangeable.

Effet sur l'environnement de rangement :
aucun effet sur l'ER actif.

4.B.3. Accès séquentiel à un objet de bureau de type déterminé

a) Donner la première occurrence d'un objet de bureau de type déterminé

Syntaxe :
donner_premier (type_objet, **ref_objet**, **code_retour**)

Objectif : fournir l'identifiant interne du premier objet de bureau de type déterminé.

Argument :
type_objet : le nom d'un type d'objet de bureau.

Contrainte :
"type_objet" identifie un type d'objet de bureau présent dans la structure de l'ER actif.

Résultats :
ref_objet : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : un objet de bureau a été trouvé;
1 : il n'y a pas d'objet du type demandé dans l'ER. actif;
41 : type d'objet inconnu de la structure de l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
"ref_objet" identifie le premier objet de bureau appartenant à l'ER actif et qui est de type "type_objet". Pour toute valeur de "code_retour" supérieure à 0', "ref_objet prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucune modification de l'ER suite à l'application de cette primitive.

b) Donner l'occurrence suivante d'un objet de bureau de type déterminé

Syntaxe :
donner_suivant (type_objet, *ref_objet*, **code_retour**)

Objectif : fournir l'identifiant interne de l'objet de bureau de type déterminé qui suit l'objet identifié par "ref_objet".

Arguments :
type_objet : le nom d'un type d'objet de bureau.
ref_objet : l'identifiant interne d'un objet de bureau.

Contraintes :
"type_objet" correspond à un type d'objet connu de la structure de l'ER actif.
"ref_objet" identifie un objet de bureau appartenant à l'ER actif et de type "type_objet".

Résultats :
ref_objet : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : un objet de bureau a été trouvé;
1 : l'objet de bureau identifié par "ref_objet" était le dernier du type considéré;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
5 : l'objet identifié par "ref_objet" n'est pas de type "type_objet";
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
41 : type d'objet inconnu de la structure de l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_objet" identifie l'objet de bureau de type "type_objet" qui suit l'objet désigné par "ref_objet". Pour toute valeur de "code_retour" supérieure à '0', "ref_objet prend la valeur 'null'.

Effet sur l'environnement de rangement :
pas de modification de l'ER actif.

Remarque :
si "ref_objet" a la valeur 'null', "donner_suivant" se comporte comme "donner_premier".

4.B.4. Accès à un objet de bureau par sa valeur d'identifiant externe

Syntaxe :
acces_direct (nom_objet, **ref_objet, code_retour**)

Objectif : trouver l'identifiant interne d'un objet de bureau par sa valeur d'identifiant externe.

Argument :
nom_objet : l'identifiant externe d'un objet de bureau.

Contrainte :
"nom_objet" est l'identifiant externe complet d'un objet de bureau appartenant à l'ER actif.

Résultats :
ref_objet : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'objet de bureau identifié par "nom_objet" a été trouvé;
1 : aucun objet de bureau n'a été trouvé dans l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
"ref_objet" est l'identifiant interne de l'objet de bureau de nom "nom_objet". Si "code_retour" a une valeur supérieure à '0', "ref_objet" prend la valeur 'null'.

Effet sur l'environnement de rangement :
pas de changement de l'ER actif suite à l'exécution de cette primitive.

4.B.5. Parcours séquentiel du contenu d'un objet de rangement

a) Accéder au premier composant d'un objet de rangement

Syntaxe :
acces_premier (ref_contenant, **ref_contenu, indicateur_localisation, code_retour**)

Objectif : donner l'identifiant interne du premier composant de l'objet de rangement identifié par "ref-contenant".

Argument :
ref_contenant : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_contenant" identifie un objet de rangement appartenant à l'ER actif.

Résultats :
ref_contenu : l'identifiant interne d'un objet de bureau.
indicateur_localisation : voir section 3.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : l'objet rangeable a été trouvé;
1 : l'objet de rangement identifié par "ref_contenant" est vide : aucun composant n'a été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
31 : valeur d'identifiant interne ("ref_contenant") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_contenu" identifie le premier composant de l'objet de rangement identifié par "ref_contenant". La valeur de "indicateur_localisation" n'est significative que pour une valeur de "code_retour" égale à '0'. Si l'objet de rangement considéré est vide ou si la valeur de "code_retour" est supérieure à '1', "ref_contenu" prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucun effet sur l'ER.

b) Accéder au composant suivant d'un objet de rangement

Syntaxe :
acces_suivant (ref_contenant, *ref_contenu*,
indicateur_localisation, code_retour)

Objectif : trouver l'identifiant interne du composant d'un objet de rangement. Ce composant suit celui qui est identifié par "ref_contenu".

Arguments :
ref_contenant, ref_contenu : des identifiants internes d'objet de bureau.

Contraintes :
"ref_contenant" identifie un objet de rangement appartenant à l'ER actif.
"ref_contenu" identifie un composant de "ref_contenant".

Résultats :
ref_contenu : l'identifiant interne d'un objet de bureau.
indicateur_localisation : voir section 3.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : l'objet rangeable a été trouvé;
1 : l'objet rangeable identifié par "ref_contenu" était le dernier composant de l'objet considéré.
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
5 : l'objet identifié par "ref_contenu" n'est pas un composant de l'objet de rangement identifié par "ref-contenant";
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
31 : valeur d'identifiant interne ("ref_contenant") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_contenu" identifie un composant qui, au sein de l'objet de rangement identifié par "ref_contenant", suit l'objet rangeable identifié par "ref_contenu". La valeur de "indicateur_localisation" ne sera significative que pour une valeur de "code_retour" égale à '0'. Pour toute valeur de "code_retour" supérieure à '0', "ref_contenu" prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucune modification de l'ER actif.

Remarque :
si "ref_contenu" a la valeur 'null', "acces_suivant" se comporte comme "acces_premier".

4.B.6. Création, suppression et modification d'un objet de bureau

a). Créer un objet dans l'environnement de rangement

Syntaxe :

creer (liste_attr, ref_contenant, **ref_contenu, code_retour**)

Objectif : créer et ranger un objet de bureau dans l'environnement de rangement actif.

Arguments :
liste_attr : l'ensemble des valeurs des attributs d'un objet de bureau.
ref_contenant : l'identifiant interne d'un objet de bureau.

Contraintes :
"ref_contenant" identifie un objet de rangement appartenant à l'ER actif, celui-ci est en mesure de contenir l'objet que l'on va créer.
Toutes les valeurs des champs de "liste_attr" sont définies sauf éventuellement les champs qui correspondent aux attributs d'un document (cela lorsque l'on crée un objet de bureau autre qu'un document). Si le champ "nomobjet" a une valeur blanche (string vide) alors c'est la primitive qui se charge de trouver un nom pour le nouvel objet de bureau.

Résultats :
ref_contenu : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de code_retour sont :

0 : l'objet a été créé et rangé dans l'ER.;
2 : un objet de bureau de ce type avec un nom identique existe déjà;
4 : l'objet identifié par ref_contenant n'a pas été trouvé;
31 : valeur d identifiant interne (ref_contenant) incorrecte;
40 : création interdite pour un objet de ce type;
41 : le type de l'objet à créer est inconnu de la structure de l'ER. actif;
42 : inclusion interdite par la structure de l'ER. actif;
43 : dépassement du nombre maximum d'occurrences permises de ce type d'objet par la structure de l'ER. actif;
50 : le fichier identifié par nom_fichier n'a pas été trouvé;
80 : ER. actif complet, plus de place
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"id_contenu" identifie l'objet que l'on vient de créer.

Effet sur l'environnement de rangement :
une nouvelle occurrence d'objet apparaît dans l'ER, ses valeurs d'attribut sont celles qui sont présentes dans "liste_attr". L'objet désigné par "ref_contenant" est l'objet de rangement (d'origine et réel) de l'objet créé. Si l'objet créé est un document, alors le contenu du fichier identifié par "nomfichier" est présent dans l'ER. Le compteur d'occurrences ("nb_occu_presentes") du type de l'objet créé est mis à jour (-1).

b) Détruire un objet appartenant à l'environnement de rangement

Syntaxe :
detruire (ref_objet, **code_retour**)

Objectif : détruire un objet de bureau.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contraintes :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif. Cet objet ne peut être de type 'BUREAU'. Si "ref_objet" identifie un objet de rangement, celui-ci doit être vide.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'objet a été détruit et retiré de l'ER.;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
40 : primitive impossible pour l'objet considéré;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
l'objet de bureau identifié par "ref_objet" est retiré de l'ER actif. Le texte libre qui lui était éventuellement associé est également détruit. L'attribut "nb_occu_presentes" du type de l'objet supprimé est mis à jour (+1).

Remarque :
si l'objet détruit est un document dont le corps n'est pas stocké dans l'ER, ce dernier n'est pas détruit.

c) Obtenir l'ensemble des caractéristiques d'un objet de bureau

Syntaxe :
caracteristiques (ref_objet, **liste_attr**, **code_retour**)

Objectif : consulter l'environnement de rangement actif pour connaître des informations concernant un objet de bureau donné à partir de son identifiant interne.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif; objet de type quelconque.

Résultats :
liste_attr : l'ensemble des valeurs des attributs d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : tout s'est bien déroulé;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
"liste_attr" reprend les valeurs des attributs de l'objet de bureau identifié par "ref_objet". Lorsque "ref_objet" identifie un objet qui n'est pas de type 'DOCUMENT', les attributs spécifiques à ce type d'objet prennent une valeur 'blanche' (un string vide pour les chaînes de caractères ou une valeur '0' pour les entiers). Pour toute valeur de "code_retour" supérieure à '0', les éléments de "liste_attr" prennent la valeur 'blanche' également.

Effet sur l'environnement de rangement :
l'application de cette primitive n'a aucun effet sur l'ER actif.

d) Modifier une ou de plusieurs caractéristiques d'un objet de bureau

Syntaxe :
modification_attributs (ref_objet, liste_attr, **code_retour**)

Objectif : mettre à jour certains attributs d'un objet de bureau à partir de "liste_attr".

Arguments :
ref_objet : l'identifiant interne d'un objet de bureau.
liste_attr : l'ensemble des valeurs des attributs d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif; objet de type quelconque.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'objet a été modifié;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
Si les modifications portent sur des attributs non accessibles par cette primitive (cfr remarque), les anciennes valeurs sont conservées. Si l'objet considéré est un objet rangeable et que l'attribut modifié est à la base d'un classement, il y a reclassement immédiat de cet objet.
De même, si l'on traite un objet de rangement et que la modification porte sur "modeclassementobjet" ou sur "attributclassementob", il y a reclassement des composants de l'objet de rangement considéré.

Effet sur l'environnement de rangement :
les attributs de l'objet identifié par "ref_objet" sont mis à jour grâce à "liste_attr". Si les modifications des attributs le demandent (modededeclassementobjet, attributclassementob, ...), il y a réorganisation des classements dans l'ER.

Remarque :
les attributs modifiables par cette primitive sont : "createurresponsable", "referenceoriginal "modeclassementobjet", "attributclassementob", "motcleobjet1", "mocleobjet2", "motdepasse" et "confidentialité".

e) Rechercher la localisation d'un objet de bureau au sein de l'environnement de rangement

- recherche de la localisation d'origine -

Syntaxe :
localisation_origine (ref_contenu, **ref_contenant, code_retour**)

Objectif : déterminer le lieu de rangement d'origine d'un objet rangeable.

Argument :
ref_contenu : l'identifiant interne d'un objet de bureau.

Contraintes :
"ref_contenu" identifie un objet rangeable appartenant à l'ER actif. Cet objet ne peut être de type 'BUREAU'.

Résultats :
ref_contenant : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la localisation d'origine a été trouvée;
1 : l'objet de bureau demandé n'a pas été trouvé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
"ref_contenant" identifie un objet de rangement qui est le lieu de rangement d'origine du composant considéré. Pour toute valeur de "code_retour" supérieure à '0', "ref_contenant" prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucun effet sur l'ER actif.

- recherche de la localisation réelle -

Syntaxe :
localisation_reelle (ref_contenu, **ref_contenant, code_retour**)

Objectif : déterminer la localisation réelle d'un objet rangeable.

Argument :
ref_contenu : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_contenu" identifie un objet rangeable appartenant à l'ER actif.

Résultats :
ref_contenant : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la localisation réelle a été trouvée;
1 : l'objet de bureau demandé n'a pas été trouvé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
40 : primitive impossible pour un objet de ce type;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
"ref_contenant" identifie un objet de rangement qui appartient à l'ER et qui est la localisation réelle du composant identifié par "ref_contenu". Pour toute valeur de "code_retour" supérieure à '0', "ref-contenant"prend la valeur 'null'.

Effet sur l'environnement de rangement :
aucune modification de l'ER actif suite à l'application de cette primitive.

f) Modifier la localisation d'un objet de bureau au sein de l'environnement de rangement

- modification de la localisation d'origine -

Syntaxe :
changer_lieu_origine (ref_contenu, ref_contenant, **code_retour**)

Objectif : modifier le lieu de rangement d'origine d'un objet de bureau.

Arguments :
ref_contenu et ref_contenant : des identifiants internes d'objet de bureau.

Contraintes :
"ref_contenu" et "ref_contenant" identifient respectivement un objet rangeable et un objet de rangement appartenant à l'ER actif; objets distincts. Les deux types d'objet auxquels correspondent respectivement les objets identifiés par "ref_contenant" et "ref_contenu" doivent partager une relation d'inclusion dans la structure de l'ER actif.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'objet de bureau a bien été déplacé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
31 : valeur d'identifiant interne ("ref-contenant") incorrecte;
40 : déplacement interdit pour un objet de ce type;
42 : inclusion interdite par la structure de l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
l'objet de rangement identifié par "ref_contenant" comporte un nouveau composant d'origine désigné par "ref_contenu". Rappelons que ce dernier ne voit pas sa localisation réelle se modifier.

- modification de la localisation réelle -

Syntaxe :
deplacer_objet (ref_contenu, ref_contenant **code_retour**)

Objectif : modifier la localisation réelle d'un objet rangeable.

Arguments :
ref_contenu, ref_contenant : des identifiants internes d'objet de bureau.

Contraintes :
"ref_contenu" identifie un objet rangeable appartenant à l'ER actif. "ref_contenant" identifie un objet de rangement appartenant à l'ER actif, celui-ci est en mesure de contenir l'objet identifié par "ref_contenu". Les deux objets doivent être distincts.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : l'objet de bureau a bien été déplacé;
3 : l'objet identifié par "ref_contenu" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_contenu") incorrecte;
31 : valeur d identifiant interne ("ref-contenant") incorrecte;
40 : déplacement interdit pour un objet de ce type;
42 :inclusion interdite par la structure de l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
l'objet de rangement identifié par "ref_contenant" comporte un nouveau composant réel désigné par "ref_contenu". Le lieu de rangement d'origine de ce dernier reste inchangé.

g) Ranger automatiquement un objet de bureau à sa place d'origine

Syntaxe :
reinserer_place_origine (ref_objet, **code_retour**)

Objectif : modifier la localisation réelle d'un objet de bureau en le rangeant effectivement à sa place d'origine.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet rangeable appartenant à l'ER actif.

Résultats :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : l'objet de bureau a été replacé à sa place d'origine;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
40 : déplacement interdit pour un objet de ce type;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
pour l'objet rangeable identifié par "ref_objet", c'est le même objet de bureau qui joue le rôle de lieu de rangement d'origine et de localisation réelle.

4.B.7. Primitives de texte libre

a) Commenter un objet de bureau

Syntaxe :
associer_texte_libre (ref_objet, nom_fichier, **code_retour**)

Objectif : associer un commentaire à un objet de bureau.

Arguments :
ref_objet : l'identifiant interne d'un objet de bureau.
nom-fichier : le nom d'un fichier externe.

Contraintes :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif; objet de type quelconque qui n'a pas encore de texte libre.
"nom_fichier" désigne un fichier externe connu.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : la primitive s'est exécutée correctement;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
50 : le fichier identifié par "nom_fichier" n'a pas été trouvé;
60 : l'objet identifié par "ref_objet" possède déjà un texte libre;
80 : ER. actif complet, plus de place;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Le fichier externe identifié par "nom_fichier" est détruit.

Effet sur l'environnement de rangement :
le contenu du fichier désigné par "nom_fichier" est présent dans l'ER actif et il est attaché à l'objet de bureau identifié par "ref_objet" en tant que texte libre.

b) Détruire le texte libre d'un objet de bureau

Syntaxe :
detruire_texte_libre (ref_objet, **code_retour**)

Objectif : détruire le texte libre d'un objet de bureau.

Argument :
ref_objet : l'identifiant interne d'un objet de bureau.

Contrainte :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif. Objet de type quelconque mais auquel est associé un texte libre.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : le texte libre a été détruit;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
61 : aucun texte libre n'est attaché à cet objet de bureau;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Effet sur l'environnement de rangement :
le texte libre attaché à l'objet de bureau identifié par "ref_objet" est retiré de l'ER actif.

c) Fournir le texte libre d'un objet de bureau

Syntaxe :
lire_texte_libre (ref_objet, nom_fichier, **code_retour**)

Objectif : fournir le texte libre d'un objet de bureau sous la forme d'un fichier externe ("nom_fichier").

Arguments :
ref_objet : l'identifiant interne d'un objet de bureau.
nom- fichier : le nom d'un fichier externe.

Contraintes :
"ref_objet" identifie un objet de bureau appartenant à l'ER actif. Objet de type quelconque mais auquel est associé un texte libre.
"nom_fichier" désigne le nom d'un fichier externe n'existant pas encore.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : la primitive s'est déroulée correctement;
3 : l'objet identifié par "ref_objet" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_objet") incorrecte;
51 : le fichier identifié par "nom_fichier" existe déjà;
61 : aucun texte libre n'est attaché à cet objet de bureau;
90 : ER. actif endommagé;
99 : incident technique au cours de l'opération.

Un fichier désigné par "nom_fichier" est accessible, son contenu équivaut au texte libre de l'objet de bureau identifié par "ref_objet".

Effet sur l'environnement de rangement :
aucun effet sur l'ER actif.

4.B.8. Primitives spécifiques aux documents

a) Dupliquer un objet de bureau de type 'DOCUMENT'

Syntaxe :
copie (ref_document, nom_objet, ref_contenant, nom_fichier,
ref_contenu, code_retour)

Objectif : copier un document.

Arguments :
ref_document et ref_contenant : des identifiants internes d'objet de bureau.
nom_objet : l'identifiant externe partiel d'un objet de bureau.
nom_fichier : le nom d'un fichier externe.

Contraintes :
"ref_document" identifie un document appartenant à l'environnement de rangement actif.
"ref_contenant" identifie un objet de rangement appartenant à l'ER actif. Il peut recevoir des objets de type 'DOCUMENT'.
"nom_objet" n'identifie encore aucun objet de bureau appartenant à l'ER actif. Si "nom_objet" a une valeur 'blanche' (chaîne vide), c'est la primitive qui se charge de trouver un nom pour la copie du document.

Résultats :
ref_contenu : l'identifiant interne d'un objet de bureau.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : la copie s'est déroulée correctement;
2 : l'identifiant interne fourni existe déjà dans l'ER. actif;
3 : l'objet identifié par "ref_document" n'a pas été trouvé;
4 : l'objet identifié par "ref_contenant" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_document") incorrecte;
31 : valeur d'identifiant interne ("ref_contenant") incorrecte;
40 : primitive impossible pour un objet de ce type;
42 : inclusion interdite par la structure de l'ER. actif;
43 : dépassement du nombre maximum d'occurrences permises de ce type d'objet par la structure de l'ER. actif;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

"ref_contenu" identifie un objet de bureau de type 'DOCUMENT'. Il est la copie exacte du document identifié par "ref_contenant" excepté pour la valeur des attributs "nomobjet", "nomfichier", "copiede", "datecopie" et pour son lieu de rangement d'origine et réel.

Effet sur l'environnement de rangement :
un nouvel document est créé dans l'ER actif.
Le compteur d'occurrences ("nb_occu_presentes") du type d'objet 'DOCUMENT' est mis à jour (+1). L'objet de rangement ("ref_contenant") possède un nouveau composant par la présence, en son sein, du document copié.

b) Fournir le corps d'un objet de bureau de type 'DOCUMENT'

Syntaxe :
obtenir_corps (ref_document, nom_fichier, **code_retour**)

Objectif : fournir le corps d'un document sous la forme d'un fichier externe ("nom_fichier").

Arguments :
ref_document : l'identifiant interne d'un objet de bureau.
nom_fichier : un nom de fichier.

Contraintes :
"ref_document" identifie un document appartenant à l'ER actif.
"nom_fichier" correspond à un fichier externe qui n'existe pas encore.
Si "nom_fichier" a une valeur 'blanche' (chaîne vide) alors la primitive prend comme nom de fichier la valeur présente dans les attributs ("nomfichier") du document considéré.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : tout s'est déroulé correctement;
3 : l'objet identifié par "ref_document" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_document") incorrecte;
40 : primitive interdite pour un objet de ce type;
51 : le fichier identifié par "nom_fichier" existe déjà;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Le fichier désigné par "nom_fichier" contiendra le corps du document identifié par "ref_document".

Effet sur l'environnement de rangement :
aucun effet sur l'ER actif.

c) Modifier le corps d'un document

Syntaxe :
remplacer_corps (ref_document, nom_fichier, **code_retour**)

Objectif : mettre à jour le corps d'un document.

Arguments :
ref_document : l'identifiant interne d'un objet de bureau.
nom_fichier : le nom d'un fichier externe.

Contraintes :
"ref_document" identifie un document qui appartient à l'ER actif.
"nom_fichier" identifie un fichier externe connu.
Si "nom_fichier" a une valeur 'blanche' (chaîne vide) alors la primitive prend comme nom de fichier celui qui est présent dans la liste des attributs ("nomfichier") du document manipulé.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : la modification a bien eu lieu;
3 : l'objet identifié par "ref_document" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_document") incorrecte;
40 : primitive impossible pour un objet de ce type;
50 : le fichier identifié par "nom_fichier" n'a pas été trouvé;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.
Le fichier externe identifié par "nom_fichier" est détruit.

Effet sur l'environnement de rangement :
le corps du document identifié par "ref_document" est remplacé par le contenu du fichier désigné par "nom_fichier".
Les attributs "datedermaj" et "nomfichier" sont mis à jour.

d) Modifier le mode de stockage d'un document

- passage à un stockage en dehors de l'ER -

Syntaxe :
placer_corps_hors_ER (ref_document, nom_fichier, **code_retour**)

Objectif : modifier le mode de stockage du corps d'un document en le déchargeant de l'environnement de rangement.

Arguments :
ref_document : l'identifiant interne d'un objet de bureau.
nom_fichier : un nom de fichier.

Contraintes :
"ref_document" identifie un document appartenant à l'ER actif. L'attribut 'stockage' de cet objet doit avoir la valeur 'oui'.
"nom_fichier" correspond à un fichier externe qui n'existe pas encore.
Si "nom_fichier" a une valeur 'blanche' (chaîne vide) alors la primitive prend comme nom de fichier la valeur présente dans l'attribut ("nomfichier) du document considéré.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : tout s'est déroulé correctement;
3 : l'objet identifié par "ref_document" n'a pas été trouvé;
30 : valeur d'identifiant interne ("ref_document") incorrecte;
40 : primitive interdite pour un objet de ce type;
51 : le fichier identifié par "nom_fichier" existe déjà;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
le corps du document identifié par "ref_document" est remplacé par le contenu de "nom_fichier". L'attribut 'stockage' du document considéré prend la valeur 'non'.

- passage à un stockage dans l'ER -

Syntaxe :
placer_corps_in_ER (ref_document, nom_fichier, **code_retour**)

Objectif : modifier le mode de stockage du corps d'un document en le chargeant dans l'environnement de rangement.

Arguments :
ref_document : l'identifiant interne d'un objet de bureau.
nom_fichier : un nom de fichier.

Contraintes :
"ref_document" identifie un document appartenant à l'ER actif. L'attribut 'stockage' de cet objet doit avoir la valeur'non'.
"nom_fichier" correspond à un fichier externe connu. Si "nom_fichier" a une valeur 'blanche' (chaîne vide) alors la primitive prend comme nom de fichier la valeur présente dans l'attribut ("nomfichier") du document considéré.

Résultat :
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :

0 : tout s'est déroulé correctement;
3 : l'objet identifié par ref_contenu n'a pas été trouvé;
30 : valeur d'identifiant interne (ref_contenu) incorrecte;
40 : primitive interdite pour un objet de ce type;
50 : le fichier identifié par nom_fichier n'existe pas;
90 : ER. endommagé;
99 : incident technique lors de l'exécution de la primitive.

Le fichier identifié par "nom_fichier" est détruit.

Effet sur l'environnement de rangement :
le corps du document identifié par "ref_document" est remplacé par le contenu de "nom_fichier". Les attributs "stockage" et "nomfichier" du document considéré prennent respectivement la valeur 'oui' et 'nom_fichier'.

4.B.9. Primitives de manipulation de l'identifiant interne

Dans la section 2, nous avons définis un nouveau type de données : le type IDENTIFIANT_INTERNE appelé DBKEY. Comme les variables de ce type ne peuvent être ni lues, ni écrites, nous proposons maintenant des primitives qui permettent respectivement de modifier et de contrôler la valeur de ces variables.

a) Donner la valeur 'null' à une variable de type DBKEY

Syntaxe :
mettre_a_null (*id_objet*, **code_retour**)

Objectif : affecter la valeur 'null' à une variable de type DBKEY.

Argument :
id_objet : une variable de type DBKEY.

Résultats :
id_objet : une variable de type DBKEY.
code_retour : le témoin d'exécution de la primitive.

Propriétés des résultats :
les valeurs possibles de "code_retour" sont :
0 : l'affectation a bien eu lieu;
99 : incident technique lors de l'exécution de la primitive.

Effet sur l'environnement de rangement :
aucune modification de l'ER actif.

b) Vérifier la valeur d'une variable de type DBKEY (fonction)

Syntaxe :
verif_idint (id_objet)

Objectif : contrôler la valeur d'un identifiant interne.

Argument :
id_objet : une variable de type DBKEY.

Résultat :
verif_idint : une variable de type booléen.

Propriétés des résultats :
"verif_idint" prend la valeur 'vrai' si "id_objet" a la valeur 'null', sinon il prend la valeur 'faux'.

Effet sur l'environnement de rangement :
aucune modification de l'ER actif.

Année académique 1988 - 1989.

Contribution à la Conception et à la Réalisation d'un SGBD d'objets bureautiques

ANNEXE II

Mémoire présenté en vue de l'obtention du diplôme
de Licence et Maître en Informatique.

Pierre HUBAR
Daniel TASSEROUL
Promoteur : R.LESUISSE

PLAN DE L'ANNEXE II

INTRODUCTION

Dans cette annexe II, nous présentons respectivement :

- le rapport de la base de données c'est-à-dire la description de la base de données, la liste des entités et celle des associations. Pour chaque type d'entité, les paramètres physiques (mode de stockage et l'intervalle de page quant le mode est Random Storage) sont précisés. Ce rapport est précédé d'un rappel concernant le schéma conforme à NDBS et des contraintes d'intégrité qui y sont associées;

- les types de données et les variables générés par le gestionnaire de schéma. Ces types et variables sont nécessaires pour manipuler les procédures du Data Base Handler et donc pour pouvoir travailler sur la base de données;

- les choix des paramètres physiques du schéma NDBS et leur justification;

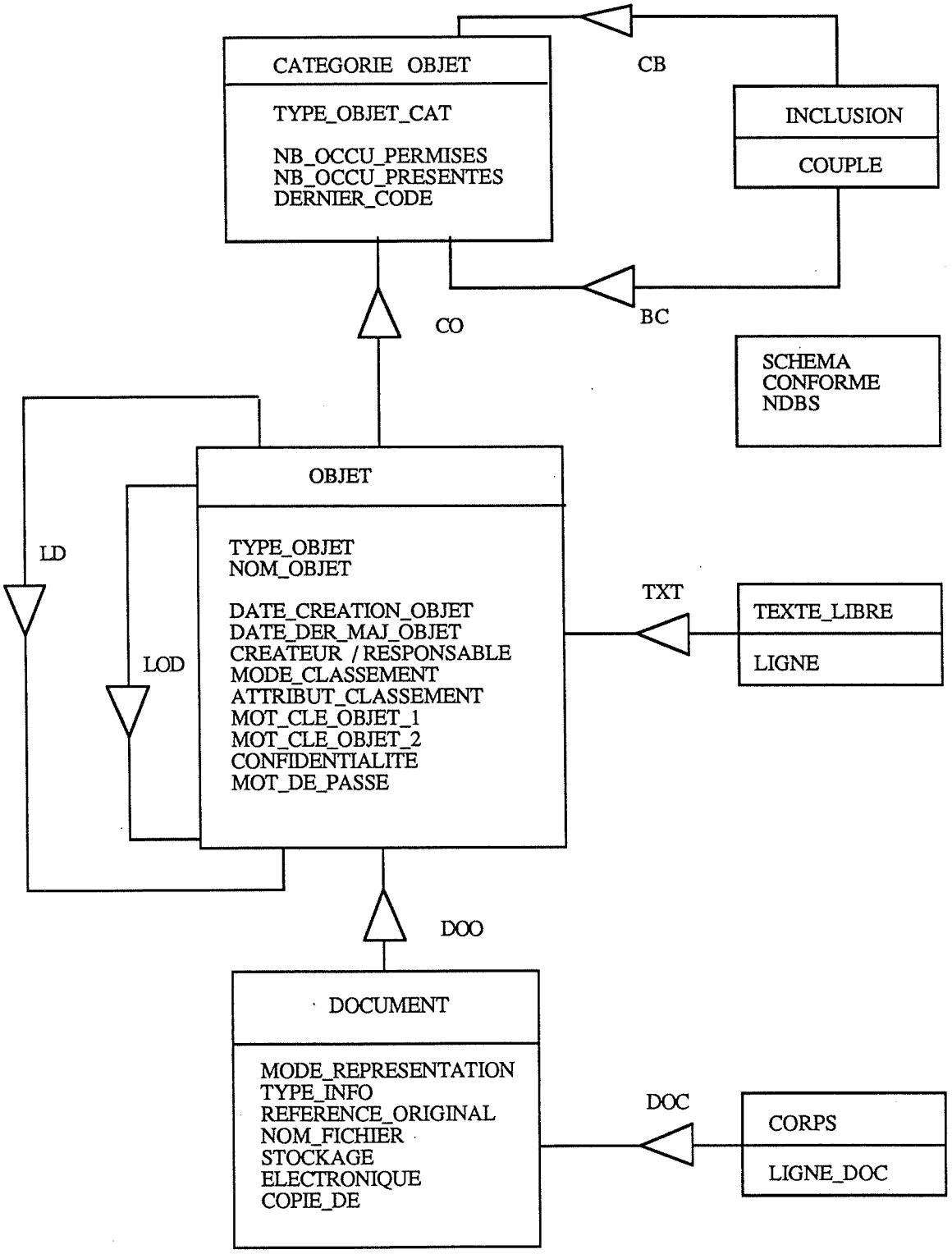
- un inventaire des primitives avec renvoi aux pages de cette annexe;

-le code-source des primitives. En plus des commentaires inclus dans ce code, le lecteur trouvera à la fin de cette annexe une feuille volante sur laquelle est repris le schéma de la base de données, ceci dans le but de l'aider à comprendre l'emploi des primitives NDBS;

- le code-source des macro-primitives présentées dans le chapitre V;

- et enfin le code-source de l'exemple d'application servant de démonstration.

1. Le schéma conforme à NDBS



CONTRAINTES D'INTEGRITE SUPPLEMENTAIRES :

- le type de chemin DO est de classe fonctionnelle 1-1; ;
- contraintes d'existence :

  - toute occurrence d'entité CORPS doit être associée à une occurrence d'entité DOCUMENT via le type de chemin DOC;

  - toute occurrence d'entité DOCUMENT doit être associée à une occurrence d'entité OBJET via le type de chemin DOO;
  - toute occurrence d'entité OBJET (sauf celle où TYPE_OBJET = bureau) doit être associée à une occurrence d'entité OBJET via le type de chemin LO et LOD;

  - toute occurrence d'entité TEXTE_LIBRE doit être associée à une occurrence d'entité OBJET via le type de chemin TXT;

  - toute occurrence d'entité OBJET doit être associée à une occurrence d'entité CATEGORIE_OBJET via le type de chemin CO;
  - toute occcurrence d'entité CATEGORIE_OBJET doit être associée à une occurrence d'entité INCLUSION via un type de chemin CB et BC; et inversément.

  - contraintes sur attributs (voir plus haut);

- contrainte référentielle :
  OBJET.TYPE_OBJET **C** CATEGORIE.TYPE_OBJET_CAT.

2. RAPPORT SUR UNE BASE DE DONNEES

****

NOM: BUREAU
Taille_Buffer: 2 à 30 K: 20
Description:
BASE DE DONNEES SERVANT DE SUPPORT AUX PRIMITIVES OFFERTES P
OUR DES APPLICATIONSEN BUREAUTIQUE

Liste des types d'Entités
==========

1 : CATEGORIEOBJET
Mode_Stockage: R 1 à 20
Description:
TYPE D'ENTITE REPRENANT L'ENSEMBLE DES TYPES D'OBJETS MANIPU
LES
Attributs:
TYPEOBJETCAT : String[15]
NBOCCUPERMISES : Integer
NBOCCUPRESENTES : Integer
DERNIERCODEATTRIBUT : Integer

Description_Attribut:
TYPEOBJETCAT :
DONNE LE TYPE DE L'OBJET
NBOCCUPERMISES :
NOMBRE MAXIMUN D'OCCURRENCES D'UN TYPE D'OBJET QUE L'ON ADME
T DANS L'E.R
NBOCCUPRESENTES :
NOMBRE D'OCCURENCES DU TYPE D'OBJET PRESENTES AU MOMENT DE L
A CONSULTATION
DERNIERCODEATTRIBUT :
FOURNIT LE DERNIER CODE GENERE POUR UN OBJET DU TYPE D'OBJET
"TYPEOBJETCAT"

2 : OBJET
Mode_Stockage: R 1 à 30000
Description:
REPRESENTE UNE OCCURRENCE D'OBJET
Attributs:
TYPEOBJET : String[15]
NOMOBJET : String[30]
DATECREATIONOBJET : record
JOUR : Integer
MOIS : Integer
ANNEE : Integer
end
DATEDERMAJOBJET : record
JOURMAJ : Integer
MOISMAJ : Integer
ANNEEMAJ : Integer
end
CREATEURRESPONSABLE : String[20]
MODECLASSEMENTOBJET : Integer
ATTRIBUTCLASSEMENTOB : Integer
MOTCLEOBJET1 : String[30]
MOTCLEOBJET2 : Integer
CONFIDENTIALITE : Boolean
MOTDEPASSE : String[20]

Description Attribut:

TYPEOBJET :

DONNE LE TYPE DE L'OBJET

NOMOBJET :

IDENTIFIANT DE L'OBJET , EST CONCU EN PRENANT LES 3 PREMIERE S LETTRES DU TYPE ETEN AJOUTANT LA VALEUR D'UN COMPTEUR PROP RE AU TYPE (DERNIERCODEATTRIBUT)

DATECREATIONOBJET :

DONNE LA DATE DE CREATION DE L'OBJET

JOUR :

MOIS :

ANNEE :

DATEDERMAJOBJET :

DONNE LA DATE DE DERNIERE MISE A JOUR DE L'OBJET

JOURMAJ :

MOISMAJ :

ANNEEMAJ :

CREATEURRESPONSABLE :

DESIGNE LA PERSONNE QUI A CREE L'OBJET OU QUI EST RESPONSABL E DE L'OBJET

MODECLASSEMENTOBJET :

PERMET DE SAVOIR SI L'OBJET EST TRIE OU NON : LA VALEUR 0 = NON TRIE1 = TRIE CROISSANT. 2 TRIE DECROISSANT

ATTRIBUTCLASSEMENTOB :

DONNE L'ATTRIBUT DE L'OBJET RANGE, ATTRIBUT BASE DU CLASSEME NT

MOTCLEOBJET1 :

ATTRIBUT DE TYPE STRING PERMETTANT UNE CLASSIFICATION SUR UN TYPE DE VALEUR QUELCONQUE

MOTCLEOBJET2 :

ATTRIBUT DE TYPE NUMERIQUE PERMETTANT UNE CLASSIFICATION SUR UNE VALEUR QUELCONQUE

CONFIDENTIALITE :

PERMET DE CONNAITRE SI L'OBJET EST CONFIDENTIEL OU NON

MOTDEPASSE :

FOURNIT LE MOT DE PASSE QUI EST, DANS L'APPLICATION, NECESSA IRE POUR ACCEDERA L'OBJET

3 : DOCUMENT

Mode Stockage: C

Description:

PRECISE L'OBJET QUANT CELUI-CI EST UN DOCUMENT

Attributs:

```
MODEREPRESENTATION : String[20]
TYPEINFO : String[20]
REFERENCEORIGINAL : String[30]
NOMFICHIER : String[50]
STOCKAGE : Boolean
COPIEDE : String[30]
DATECOPIE : record
              JOURCOPIE : Integer
              MOISCOPIE : Integer
              ANNEECOPIE : Integer
            end
ELECTRONIQUE : Boolean
```

Description_Attribut:
MODEREPRESENTATION :
DONNE LE MODE DE REPRESENTATION DU DOCUMENT
TYPEINFO :
DONNE LE TYPE DU DOCUMENT (LETTRE, MEMOIRE,...)
REFERENCEORIGINAL :
DONNE LA REFERENCE DE L'ORIGINAL SI LE DOCUMENT EN EST UNE C
OPIE
NOMFICHIER :
DONNE LE NOM DU FICHIER PLUS EVENTUELLEMENT LE CHEMIN D'ACCE
S, SOUS LEQUEL LE DOCUMENT SERA RESTITUE HORS DE LA BASE DE
DONNEES
STOCKAGE :
PERMET DE SAVOIR SI LE CORPS DU DOCUMENT EST STOCKE EN BASE
DE DONNEES
COPIEDE :
DONNE LE NOM DE L'OBJET QUI A ETE COPIE
DATECOPIE :
DONNE LA DATE A LAQUELLE LE DOCUMENT CONCERNE A ETE CREE PAR
COPIE
JOURCOPIE :
MOISCOPIE :
ANNEECOPIE :
ELECTRONIQUE :
PERMET DE SAVOIR SI LE DOCUMENT EST DE NATURE ELECTRONIQUE

4 : TEXTELIBRE
Mode_Stockage: C
Description:
PERMET DE STOCKER LE TEXTE LIBRE PRECISANT DE FACON NON STAN
DARD UN OBJET
Attributs:
LIGNE : String[254]

Description_Attribut:
LIGNE :
SERT AU STOCKAGE DU TEXTE LIBRE

5 : CORPS
Mode_Stockage: C
Description:
PERMET LE STOCKAGE DU CORPS DU DOCUMENT
Attributs:
LIGNEDOC : String[254]

Description_Attribut:
LIGNEDOC :
PERMET LE STOCKAGE DU CORPS DU DOCUMENT SOUS FORME DE CHAINE
DE CARACTERES

6 : INCLUSION
Mode_Stockage: C
Description:
ENTITE REPRESENTANT LES DIVERSES INCLUSIONS LOGIQUES ENTRE L
ES TYPES D'OBJET
Attributs:
COUPLE : String[31]

Description_Attribut:
COUPLE :
REDONDANCE QUI PERMET DE REPRESENTER LES RELATIONS D'INCLUSI
ON

Liste des Types d'Association
==========

1 : CO

Origine: CATEGORIEOBJET
Cible: OBJET
Description:

2 : LO

Origine: OBJET
Cible: OBJET
Description:

3 : LOD

Origine: OBJET

Cible: OBJET
Description:

4 : TXT

Origine: OBJET
Cible: TEXTELIBRE
Description:

5 : DOO

Origine: OBJET
Cible: DOCUMENT
Description:

6 : DOC

Origine: DOCUMENT
Cible: CORPS
Description:

7 : CB

Origine: CATEGORIEOBJET
Cible: INCLUSION
Description:

8 : BC

Origine: CATEGORIEOBJET
Cible: INCLUSION
Description:

3. LES TYPES DE DONNEES ET LES VARIABLES

```
UNIT TYPBUR;
INTERFACE

CONST
     XXLKEY = 3 ;
     XXNBETE = 6 ;
     XXNBETA = 8 ;
     XXLGBUF = 20 ;
     XXLGPAGE = 1024 ;
     XXLGBLOC = 4 ;
     XXMAXCAR = 260 ;
     XXNBIND1 = 256 ;  (* XXLGPAGE / 4 *)
     XXNBIND2 = 512 ;  (* XXLGPAGE / 2 *)

     CATEGORIEOBJET = 1 ;
     OBJET = 2 ;
     DOCUMENT = 3 ;
     TEXTELIBRE = 4 ;
     CORPS = 5 ;
     INCLUSION = 6 ;

     CO = 1 ;
     LO = 2 ;
     LOD = 3 ;
     TXT = 4 ;
     DOO = 5 ;
     DOC = 6 ;
     CB = 7 ;
     BC = 8 ;

TYPE
     XXDESCTE = record
                  LgItem : byte ;
                  LgPoint : byte ;
                  PosId : byte ;
                  LgId : byte ;
                  NatId : byte ;
                  Pd1 : byte ;
                  Pd2 : byte ;
                  Pf1 : byte ;
                  Pf2 : byte ;
                end;

     XXDESCTA  = record
                   TypOrig : byte ;
                   TypCib : byte ;
                   PositOrig : byte ;
                   PositCib : byte ;
                 end;

     XXTABTE = array [1..XXNBETE] of XXDESCTE ;
     XXTABTA = array [1..XXNBETA] of XXDESCTA ;
```

```
XXKEY = array [1..XXLKEY] of char ;
DBKEY = XXKEY ;
XXTABINDIR = array [1..XXNBETE] of XXKEY ;

XXINDEX = record
            place : integer ;
            ptr : integer ;
          end;
XXTABINDEX = array [1..XXNBIND1] of XXINDEX ;

XXPAGE = array [1..XXLGPAGE] of char ;
XXLIGNE = record
            NumPage : integer ;
            Date : integer ;
            Modif : boolean ;
          end;
XXTAMPON = record
             Cadre : array [1..XXLGBUF] of XXPAGE ;
             Descripteur : array [1..XXLGBUF] of XXLIGNE
             ;
           end;

TCATEGORIEOBJET = record
          xxref : XXKEY ;
          xxcode : char ;
          TYPEOBJETCAT : String[15] ;
          NBOCCUPERMISES : Integer ;
          NBOCCUPRESENTES : Integer ;
          DERNIERCODEATTRIBUT : Integer ;
        end ;

TOBJET = record
          xxref : XXKEY ;
          xxcode : char ;
          TYPEOBJET : String[15] ;
          NOMOBJET : String[30] ;
          DATECREATIONOBJET : record
                                JOUR : Integer ;
                                MOIS : Integer ;
                                ANNEE : Integer ;
                              end;
          DATEDERMAJOBJET : record
                              JOURMAJ : Integer ;
                              MOISMAJ : Integer ;
                              ANNEEMAJ : Integer ;
                            end;
          CREATEURRESPONSABLE : String[20] ;
          MODECLASSEMENTOBJET : Integer ;
          ATTRIBUTCLASSEMENTOB : Integer ;
          MOTCLEOBJET1 : String[30] ;
          MOTCLEOBJET2 : Integer ;
          CONFIDENTIALITE : Boolean ;
          MOTDEPASSE : String[20] ;
        end ;
```

```
    TDOCUMENT = record
              xxref : XXKEY ;
              xxcode : char ;
              MODEREPRESENTATION : String[20] ;
              TYPEINFO : String[20] ;
              REFERENCEORIGINAL : String[30] ;
              NOMFICHIER : String[50] ;
              STOCKAGE : Boolean ;
              COPIEDE : String[30] ;
              DATECOPIE : record
                            JOURCOPIE : Integer ;
                            MOISCOPIE : Integer ;
                            ANNEECOPIE : Integer ;
                          end;
              ELECTRONIQUE : Boolean ;
            end ;

    TTEXTELIBRE = record
              xxref : XXKEY ;
              xxcode : char ;
              LIGNE : String[254] ;
            end ;

    TCORPS = record
              xxref : XXKEY ;
              xxcode : char ;
              LIGNEDOC : String[254] ;
            end ;

    TINCLUSION = record
              xxref : XXKEY ;
              xxcode : char ;
              COUPLE : String[31] ;
            end ;

IMPLEMENTATION
END.
```

```
UNIT DECLAR;

INTERFACE

USES TYPBUR, DBMS;

TYPE TYPEOBJET = STRING [15];
     NOMOBJET = string [30];
     NOMFICHIER = string [14];
     NOUVEAUTYPE = record
                        type_objet : string [15];
                        nb_occu_permises : integer;
                        nb_occu_presentes : integer;
                        dernier_code : integer;
                   end;

     NOM_DE_ER = string [64];
     OBJET_ATTR = record
                       ref_objet : DBKEY;
                       typeobjet : string [15];
                       nomobjet  : string [30];
                       datecreationobjet :
                             record
                                   jour : integer;
                                   mois : integer;
                                   annee : integer;
                             end;
                       datedermaj :
                              record
                                    jourmaj : integer;
                                    moismaj : integer;
                                    anneemaj: integer;
                              end;
                       createurresponsable : string [20];
                       modeclassementobjet : integer;
                       attributclassementob : integer;
                       motcleobjet1 : string[30];
                       motcleobjet2 : integer;
                       confidentialite : boolean;
                       motdepasse : string [20];
                       moderepresentation : string [20];
                       typeinfo : string [20];
                       referenceoriginal : string [30];
                       nomfichier : string [50];
                       stockage : boolean;
                       copieok : string [30];
                       datecopie :
                            record
                                  jourcopie : integer;
                                  moiscopie : integer;
                                  annecopie : integer;
                                  end;
                        electronique : boolean;
                  end ;
```

```
      cellule = ^element;
      element = record
                      ref_objet : DBKEY;
                      chaine : string [30];
                      next : cellule
                end;
       TYPSTR = string [30];
       NOMATTRIBUT = STRING[20];

VAR code_retour : integer;
    type_objet : TYPEOBJET;
    typeobjet_rg, typeobjet_rb : TYPEOBJET;

    ref_copie,    ref_objet : DBKEY;
    nom_objet : NOMOBJET;
    ref_suivant, ref_contenu , ref_contenant : DBKEY;
    continue : char;

    nouv_type : NOUVEAUTYPE;
    nouvelle_val_entier : integer;


    nom_er : NOM_DE_ER;
    objet_car ,objet_a_creer : OBJET_ATTR;
    confidentialite, stockage : boolean;
    code_localisation : integer;
    lieu : char;

    annee , mois , jour : integer;
    nom_fichier : NOMFICHIER;

IMPLEMENTATION
END.
```

4. Les paramètres physiques

Dans la version de NDBS disponible, nous pouvons manipuler les paramètres physiques suivants :

- la taille du buffer,
- le mode de stockage d'une entité et ...
- l'intervalle de page si le mode est Random Storage.

Il en résulte que la taille des pages est celle qui est fixée par défaut à savoir: 1024 octets. Quant à la taille du buffer, elle est fixée arbitrairement à 20 K. Il faudrait des tests méthodiques pour en fixer la taille optimale.

Nos primitives manipulent des objets (types abstraits). Aussi, dans un but d'optimisation, il nous semble utile de regrouper les entités représentant un objet dans des pages contigues (cela se fera à la création d'un objet et autour de l'entité OBJET). Dès lors, le mode de stockage des entités DOCUMENT, TEXTELIBRE et CORPS est le mode Clustered. Ce choix est d'autant plus justifié pour les entités TEXTELIBRE et CORPS que celles-ci représentent, à une ou plusieurs, un texte qu'il faut stocker et restituer (la lecture sera plus rapide si chacune des entités est contiguë à l'autre). Pour pouvoir respecter ce regroupement, le mode de stockage de l'entité OBJET est le mode Random et l'intervalle de page est le maximun que permet la base de données.

Nous appliquons le même principe pour les objets (type abstrait) catégorie objets de bureau. Ainsi, nous choisissons le mode Random pour les entités CATEGORIEOBJET et le mode Clustered pour les entités INCLUSION. Nous avons limité l'intervalle de page des entités CATEGORIEOBJET à 20 pages car travailler avec plus de 20 types d'objets différents semble difficilement concevable.

5 RÉSUMÉ DES PRIMITIVES

5.A. LES PRIMITIVES D'INITIALISATION DE L'ENVIRONNEMENT DE RANGEMENT

5.B LES PRIMITIVES OUTILS

5.C. LES PRIMITIVES DE CONSTRUCTION ET DE MISE À JOUR DE LA STRUCTURE DE L'ENVIRONNEMENT DE RANGEMENT

5.D. LES PRIMITIVES D'ACCES AUX OBJETS DE BUREAU

* *
*

6. Le code-source des primitives

```
UNIT INITIALI;

INTERFACE

USES DOS, TYPBUR, DBMS, DECLAR, OUTIL;

procedure creer_er  (nom_ER : nom_de_er;
                     var code_retour : integer);

procedure ouvrir_er (nom_er : NOM_DE_ER ;
                     var code_retour : integer);

procedure fermer_er (nom_er : NOM_DE_ER ;
                     var code_retour : integer);

IMPLEMENTATION


procedure creer_er

label sortie;

var
  fich, fich1 : file of char;
  car : char;

begin
  code_retour := 0;
  assign(fich, 'ERINIT');
  {$I-} reset(fich) {$I+};
  if (IOresult = 0)
    then
      begin
        assign(fich1, nom_ER+'.dtb');
        {$I-} reset(fich1) {$I+};
        if (IOresult <> 0)
          then
            begin
              rewrite(fich1); read(fich, car);
              while not eof(fich) do
                begin
                  write(fich1, car);
                  read(fich,car);
                end;
                write(fich1,car);
            end
          else code_retour := 2;
      end
    else code_retour := 1;
  sortie : ;
end;
```

```
procedure ouvrir_er ;

begin
     dbopen (nom_er);
     code_retour := dbstatus;
end;

procedure fermer_er ;

begin
     dbclose;
     code_retour := dbstatus;
end;
END.
```

```
UNIT OUTIL;

INTERFACE

USES DOS,TYPBUR, DBMS, DECLAR ;

procedure Date (var annee , mois, jour : integer);

procedure calcul_localisation (ref_contenu: DBKEY;
                               var code_localisation : integer;
                               var code_retour : integer);

procedure maj_date_maj_contenant ( ref_contenant : DBKEY ;
                               var code_retour : integer);

procedure lecture_contenus (ref_contenant : DBKEY;
                                        lieu : char;
                             var tete : cellule;
                             var code_retour : integer);

procedure inclure_element_string_croissant
                                        (val_string : TYPSTR;
                                         ref_contenu : DBKEY;
                                         var tete : cellule);

procedure inclure_element_string_decroissant
                                        (val_string : TYPSTR;
                                        ref_contenu : DBKEY;
                                           var tete : cellule);

procedure inserer_nouveau_element ( ref_contenu : DBKEY;
                                              lieu : char;
                                    ref_contenant : DBKEY;
                                   var code_retour : integer);

procedure inverser_liste (ref_objet : DBKEY; lieu : char;
                                     code_retour : integer);

procedure trie (ref_objet : DBKEY; indicateur :integer;
                                  var code_retour : integer);

procedure decoupe_fichier ( ref_docu : DBKEY;
                                    nom_fichier : NOMFICHIER;
                                    r code_retour : integer);

procedure sortie_fichier (nom_fichier : NOMFICHIER ;
                                       ref_docu :DBKEY;
                                 var code_retour : integer);

procedure detruire_corps (ref_objet : DBKEY ;
                               var code_retour : integer );
```

```
IMPLEMENTATION

procedure Date ; {permet d'obtenir la date du système}

var
    aaaa, mm, jj, hh : word;

begin
    getdate(aaaa,mm,jj,hh);
    annee := aaaa;
    mois := mm;
    jour := jj;
end;


procedure calcul_localisation ;

 {donne la localisation de l'objet :}
 { 0 : le contenant est le lieu réel et d'origine de}
       { l'objet}
 { 1 : le contenant n'est que le lieu d'origine}
 { 2 : le contenant n'est que le lieu réel}

label exit;

var loc_origine, loc_reel : boolean;
    CONTENANT, CONTENU : TOBJET;

begin
     { calcul du code de localisation : }
     {si avec dbtestpath .... }
     {on recherche l'objet}
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 90 then goto exit;

     {on recherche le contenant lieu d'origine}
     dbfpath (CONTENANT, CONTENU,-LOD);
     if dbstatus >= 90 then goto exit;
     if dbfound then loc_origine := true
                else loc_origine := false;

     {on recherche le contenant lieu réel}
     dbfpath (CONTENANT, CONTENU, -LO);
     if dbstatus >= 90 then goto exit;
     if dbfound then loc_reel := true
                else loc_reel := false;

     if (loc_origine = true) and (loc_reel = true)
        then
            code_localisation := 0
         else
             begin
        if loc_origine = true then code_localisation := 1;
        if  loc_reel = true then code_localisation := 2;
             end;
```

```
    code_retour := 0;
    exit: ;
    if dbstatus > 10 then code_retour := dbstatus;
end;

procedure maj_date_maj_contenant ;

{permet la mise à jour de la date de dernière mise à jour}
{ d'un objet}

label exit;
var CONTENANT : TOBJET;

begin
      dbdirect (OBJET, CONTENANT, ref_contenant);
      if dbstatus >= 90 then goto exit;
      date (annee, mois, jour);
      CONTENANT.DATEDERMAJOBJET.JOURMAJ := jour;
      CONTENANT.DATEDERMAJOBJET.MOISMAJ := mois;
      CONTENANT.DATEDERMAJOBJET.ANNEEMAJ := annee;
      dbmodify (OBJET, CONTENANT);
      if dbstatus >= 80 then goto exit;
      code_retour := 0;
      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;
end;

procedure lecture_contenus ;

{permet de "lire" les contenus d'un objet. Pour chacun de}
{ ces derniers on place dans une liste chainee la }
{variable de référence et la valeur de l'attribut sur}
{laquelle sont classés ces contenus. Le résultat est}
{le pointeur de t^ete de la liste chainée}

label exit;
var p , queue : cellule;
    an, mo, jo : integer;
    val_string : TYPSTR;
    val_nombre : real;
    CONTENANT, CONTENU :TOBJET;

begin
     {on recherche le contenant}
     dbdirect (OBJET, CONTENANT, ref_contenant);
     if dbstatus >= 90 then goto exit;

     {on distingue les contenus dont le lieu d'origine}
     { ("o") est le contenant des contenus dont le}
     { contenant est le lieu réel ("r")}
     if lieu = 'r' then  dbfpath (CONTENU, CONTENANT, LO );
     if dbstatus >= 90 then goto exit;
```

```
      if lieu = 'o' then  dbfpath (CONTENU, CONTENANT, LOD );
      if dbstatus >= 90 then goto exit;

      {initialisation de la liste chainée}
      tete := nil;
      queue := nil;
      while dbfound do
        begin
        {on insère dans une liste qui au départ est vide }
          new (p);
          p^.ref_objet := CONTENU.xxref;
          case CONTENANT.ATTRIBUTCLASSEMENTOB of
           1 : p^.chaine := CONTENU.TYPEOBJET;
           2 : P^.chaine := CONTENU.NOMOBJET;
           3 : begin
               an := CONTENU.DATECREATIONOBJET.ANNEE * 10000;
               mo := CONTENU.DATECREATIONOBJET.MOIS * 1000 ;
               jo := CONTENU.DATECREATIONOBJET.JOUR ;
               val_nombre := an + mo + jo ;
               Str (val_nombre , p^.chaine);
               end;
           4 : p^.chaine := CONTENU.CREATEURRESPONSABLE ;
           5 : begin
               an := CONTENU.DATEDERMAJOBJET.ANNEEMAJ * 10000;
               mo := CONTENU.DATEDERMAJOBJET.MOISMAJ * 100 ;
               jo := CONTENU.DATEDERMAJOBJET.JOURMAJ ;
               val_nombre := an + mo + jo ;
               Str (val_nombre , p^.chaine);
               end;
           6 : p^.chaine := CONTENU.MOTCLEOBJET1;
           7 : begin
                   val_nombre := CONTENU.MOTCLEOBJET2;
                   Str (val_nombre , p^.chaine);
               end;
          end;
          p^.next := nil;
          if tete = nil then tete := p
                        else queue^.next := p;
          queue := p;
          if lieu = 'r' then
             dbnpath (CONTENU, CONTENANT, LO );
          if dbstatus >= 90 then goto exit;
          if lieu = 'o' then
             dbnpath (CONTENU, CONTENANT, LOD );
          if dbstatus >= 90 then goto exit;

        end; {lecture}

      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;

end;
```

```
procedure inclure_element_string_croissant ;

{permet d'inclure dans une liste chainée un élément}
{ en respectant un ordre croissant}
var preced ,p, r : cellule;

begin
     p := tete;
     while (p <> nil) and (p^.chaine <= val_string) do
        begin
             preced := p;
             p := p^.next;
        end;
     new (r);
     r^.ref_objet := ref_contenu;
     r^.chaine := val_string;
     r^.next := p;
     if p = tete then tete := r
                 else preced^.next := r;
end;


procedure inclure_element_string_decroissant ;

{permet d'inclure dans une liste chainée un élément}
{ en respectant un ordre décroissant}

var preced ,p, r : cellule;

begin
     p := tete;
     while (p <> nil) and (p^.chaine >= val_string) do
        begin
             preced := p;
             p := p^.next;
        end;
     new (r);
     r^.ref_objet := ref_contenu;
     r^.chaine := val_string;
     r^.next := p;
     if p = tete then tete := r
                 else preced^.next := r;
end;


procedure inserer_nouveau_element ;

{permet d'insérer une entité dans la suite d'entités}
{cibles reliées à une entité origine en respectant}
{un ordre . Cet outil sert pour un contenant}
{lieu d'origine ou lieu réel}

label exit;
```

```
var p, tete , queue : cellule;
    an, mo, jo : integer;
    val_string : TYPSTR;
    val_nombre : real;
    CONTENANT, CONTENU,CONT :TOBJET;

begin
     {on recherche le contenant}
     dbdirect (OBJET, CONTENANT, ref_contenant);
     if dbstatus >= 90 then goto exit;

     {on recherche le contenu à insérer dans les contenus}
     {déjà classés}
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 90 then goto exit;

     { on recherche un contenu du contenant ; s'il n'y}
     {en a pas on insert le nouveau contenu de suite et }
     {on procede de meme si le mode de }
     {clasement est non trie }

     if lieu = 'r' then  dbfpath (CONT, CONTENANT, LO );
     if dbstatus >= 90 then goto exit;
     if lieu = 'o' then  dbfpath (CONT, CONTENANT, LOD );
     if dbstatus >= 90 then goto exit;

    if (not dbfound) or (CONTENANT.MODECLASSEMENTOBJET = 0)
      then
        begin
    if lieu = 'r' then  dbinsert (CONTENU, CONTENANT,LO );
    if dbstatus >= 90 then goto exit;
    if lieu = 'o' then  dbinsert (CONTENU, CONTENANT,LOD );
    if dbstatus >= 90 then goto exit;

    maj_date_maj_contenant (CONTENANT.xxref, code_retour);
    if code_retour > 0 then goto exit;

    code_retour := 0;
    goto exit ;
        end;

    { on procede à la lecture des contenus et on met}
    { ceux-ci dans une liste chainee qui est triée}
    { (les contenus présents sont déjà triés)}

    lecture_contenus
              (ref_contenant, lieu, tete, code_retour);
    if code_retour > 0 then goto exit;

    {on détache les entités cibles de l'entité origine}
    p := tete;
    while p <> nil do
           begin
                dbdirect (OBJET, CONT, p^.ref_objet);
                if dbstatus >= 90 then goto exit;
```

```
            if lieu = 'r' then
                  dbremove (CONT, CONTENANT, LO );
            if dbstatus >= 90 then goto exit;
            if lieu = 'o' then
                  dbremove (CONT, CONTENANT, LOD );
            if dbstatus >= 90 then goto exit;
            p := p^.next;
       end;
{on recherche la valeur de l'attribut base de }
{classement du contenu et sa variable de référence}
if CONTENANT.ATTRIBUTCLASSEMENTOB in [1,2,4,6] then
    begin
         case CONTENANT.ATTRIBUTCLASSEMENTOB of
          1 : val_string := CONTENU.TYPEOBJET;
          2 : val_string := CONTENU.NOMOBJET;
          4 : val_string := CONTENU.CREATEURRESPONSABLE;
          6 : val_string := CONTENU.MOTCLEOBJET1;
         end;

{suivant le mode de classement du contenant on }
{inclut l'entite}
if CONTENANT.MODECLASSEMENTOBJET = 1
      then
           inclure_element_string_croissant
                (val_string ,ref_contenu, tete);
if CONTENANT.MODECLASSEMENTOBJET = 2
      then
          inclure_element_string_decroissant
           (val_string ,ref_contenu, tete);
          end;

 {on procéde de meme pour les attributs numériques}
 {dont les valeurs sont converties en string}
 if CONTENANT.ATTRIBUTCLASSEMENTOB in [3,5,7] then
    begin
    case CONTENANT.ATTRIBUTCLASSEMENTOB of
      3 : begin
          an := CONTENU.DATECREATIONOBJET.ANNEE * 10000;
          mo := CONTENU.DATECREATIONOBJET.MOIS * 1000 ;
          jo := CONTENU.DATECREATIONOBJET.JOUR ;
          val_nombre := an + mo + jo ;
          Str (val_nombre, val_string);
          end;
       5 : begin
           an := CONTENU.DATEDERMAJOBJET.ANNEEMAJ * 10000;
           mo := CONT.DATEDERMAJOBJET.MOISMAJ * 1000 ;
           jo := CONT.DATEDERMAJOBJET.JOURMAJ ;
           val_nombre  := an + mo + jo ;
           Str (val_nombre, val_string);
           end;
       7 : begin
           val_nombre := CONTENU.MOTCLEOBJET2;
           Str (val_nombre, val_string);
           end;
    end;
```

```
      if CONTENANT.MODECLASSEMENTOBJET = 1
         then
             inclure_element_string_croissant
              (val_string ,ref_contenu, tete);
      if CONTENANT.MODECLASSEMENTOBJET = 2
         then
             inclure_element_string_decroissant
              (val_string ,ref_contenu,tete);

      end;


      { réinsertion des contenus plus le nouveau contenu }

      p := tete;
      while p <> nil do
            begin
                 dbdirect (OBJET, CONT, p^.ref_objet);
                 if lieu = 'r' then
                    dbinsert (CONT, CONTENANT, LO );
                 if dbstatus >= 90 then goto exit;
                 if lieu = 'o' then
                    dbinsert (CONT, CONTENANT, LOD );
                 if dbstatus >= 90 then goto exit;
                 p := p^.next;
            end;

      maj_date_maj_contenant (CONTENANT.xxref, code_retour);
      if code_retour > 0 then goto exit;

      dbcommit;
      code_retour := 0;
      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


procedure inverser_liste ;

{permet d'inverser une liste chainée . Cet outil est}
{utilisé lorsquele mode de classement du contenant}
{passe de 1 à 2 ou de 2 à 1 }

label exit;


var tete, queue, p ,q,r : cellule;
    CONT, CONTENANT : TOBJET;
    an, jo ,mo : real;
    val_nombre : real;

begin
     dbdirect (OBJET, CONTENANT, ref_objet);
     if dbstatus >= 90 then goto exit;
```

```
if lieu = 'r' then  dbfpath (CONT, CONTENANT, LO );
if dbstatus >= 90 then goto exit;
if lieu = 'o' then  dbfpath (CONT, CONTENANT, LOD );
if dbstatus >= 90 then goto exit;

{on teste s'il y a au moins un contenu}
if not dbfound then
                    begin
                          code_retour := 0;
                          goto exit;
                    end;

{on lit les contenus}
lecture_contenus (ref_objet, lieu, tete,code_retour);
if code_retour > 0  then goto exit;

{on détache les entités cibles}
p := tete;
while p <> nil do
      begin
           dbdirect (OBJET, CONT, p^.ref_objet);
           if dbstatus >= 90 then goto exit;
           if lieu = 'r' then
              dbremove (CONT, CONTENANT, LO );
           if dbstatus >= 90 then goto exit;
           if lieu = 'o' then
              dbremove (CONT, CONTENANT, LOD );
           if dbstatus >= 90 then goto exit;
           p := p^.next;
      end;

{on inverse la liste chainée obtenue}
p := tete;
if p <> nil then
         begin
              q := nil;
              while p^.next <> nil do
                 begin
                      r := p^.next;
                      p^.next := q;
                      q :=p;
                      p := r;
                 end;
              p^.next := q;
              tete := p;
         end;

{on relie les entités cibles à l'entité origine càd}
{le contenant}

p := tete;
while p <> nil do
      begin
           dbdirect (OBJET, CONT, p^.ref_objet);
           if dbstatus >= 90 then goto exit;
```

```
                if lieu = 'r' then
                   dbinsert (CONT, CONTENANT, LO );
                if dbstatus >= 90 then goto exit;
                if lieu = 'o' then
                   dbinsert (CONT, CONTENANT, LOD );
                if dbstatus >= 90 then goto exit;
                p := p^.next;
            end;

        dbcommit;
        code_retour := 0;
        exit : ;
        if dbstatus > 10 then code_retour := dbstatus;
  end;

procedure trie ;

{outil qui permet de trie une liste chainée}

label exit;
var p, tete, tete2 : cellule;
    CONTENANT, CONT : TOBJET;

begin
     {on recherche le contenant}
     dbdirect (OBJET, CONTENANT, ref_objet);
     if dbstatus >= 90 then goto exit;

     {on trie d'abord les contenus dont le contenant}
     {est le lieu réel}
     lieu := 'r';
     lecture_contenus (ref_objet, lieu, tete,code_retour);
     if code_retour > 0 then goto exit;

     {on détache les entités cibles de leur entité origine}
     p := tete;
     while p <> nil do
          begin
               dbdirect (OBJET, CONT, p^.ref_objet);
               if dbstatus >= 90 then goto exit;
               if lieu = 'r' then
                  dbremove (CONT, CONTENANT, LO );
               if dbstatus >= 90 then goto exit;
               if lieu = 'o' then
                  dbremove (CONT, CONTENANT, LOD );
               if dbstatus >= 90 then goto exit;
               p := p^.next;
          end;

     {le tri se fait en construisant une deuxième}
     {liste chainée ( tete2) dans laquelle on inclut}
     {élement par élément }
```

```
tete2 := nil;
p := tete;
while p <> nil do
   begin
        if indicateur = 1 then
        inclure_element_string_croissant
                  (p^.chaine,p^.ref_objet,tete2);
        if indicateur = 2 then
        inclure_element_string_decroissant
                  (p^.chaine,p^.ref_objet,tete2);
        p := p^.next;
   end;

{on relie les entités cibles à l'entité origine}
{(contenant)}
p := tete2;
  while p <> nil do
      begin
           dbdirect (OBJET, CONT, p^.ref_objet);
           if dbstatus >= 90 then goto exit;
           if lieu = 'r' then
              dbinsert (CONT, CONTENANT, LO );
           if dbstatus >= 90 then goto exit;
           if lieu = 'o' then
              dbinsert (CONT, CONTENANT, LOD );
           if dbstatus >= 90 then goto exit;
           p := p^.next;
      end;

{on trie les contenus dont le contenant est}
{le lieu d'origine}
lieu := 'o';
lecture_contenus (ref_objet, lieu, tete,code_retour);
if code_retour > 0 then goto exit;

p := tete;
while p <> nil do
      begin
           dbdirect (OBJET, CONT, p^.ref_objet);
           if dbstatus >= 90 then goto exit;
           if lieu = 'r' then
              dbremove (CONT, CONTENANT, LO );
           if dbstatus >= 90 then goto exit;
           if lieu = 'o' then
              dbremove (CONT, CONTENANT, LOD );
           if dbstatus >= 90 then goto exit;
           p := p^.next;
      end;

tete2 := nil;
p := tete;
```

```
      while p <> nil do
         begin
              if indicateur = 1 then
              inclure_element_string_croissant
                      (p^.chaine,p^.ref_objet,tete2);
              if indicateur = 2 then
              inclure_element_string_decroissant
                      (p^.chaine,p^.ref_objet,tete2);
              p := p^.next;
         end;

      p := tete2;
        while p <> nil do
            begin
                 dbdirect (OBJET, CONT, p^.ref_objet);
                 if dbstatus >= 90 then goto exit;
                 if lieu = 'r' then
                    dbinsert (CONT, CONTENANT, LO );
                 if dbstatus >= 90 then goto exit;
                 if lieu = 'o' then
                    dbinsert (CONT, CONTENANT, LOD );
                 if dbstatus >= 90 then goto exit;
                 p := p^.next;
            end;


       dbcommit;
       code_retour := 0;
       exit : ;
       if dbstatus > 10 then code_retour := dbstatus;

 end;

procedure decoupe_fichier ;

{permet de stocker un fichier dans la BD sous la forme }
{d'entités dont l'unique attribut est un string de 254}
{caractères. Toutes ces entités sont reliées à }
{l'entité document}

label exit;

type fi = string [254];

var CO : TCORPS;
    DOCU : TDOCUMENT;
    f : file of char;
    ligne : fi;
    i,j : integer;
    caractere : char;


begin
       dbdirect (DOCUMENT,DOCU,ref_docu);
       if dbstatus >= 90 then goto exit;
```

```
      {on stocke le fichier}
      assign (f,nom_fichier);
      reset (f);
      while ( eof (f) = false) do
        begin
             i := 1;
             ligne := '';
             while (i <= 254) and (not eof (f))
               do
                 begin
                      read (f,caractere);
                      ligne  := ligne + caractere;
                      i := i+1;
                 end;

             CO.LIGNEDOC := ligne;
             dbcreate (CORPS,CO);
             if dbstatus >= 90 then goto exit;

             dbinsert (CO,DOCU,DOC);
             if dbstatus >= 90 then goto exit;
        end;
      close (f);
      dbcommit;
      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
   end;


procedure sortie_fichier ;

{permet de sortir une copie du corps du document}
{ou de sa référence sous forme d'un fichier}

label exit;

type li = string [254];

var CO : TCORPS;
    DOCU : TDOCUMENT;
    f : file of char;
    ligne : li;
    i : integer;

begin
      dbdirect (DOCUMENT,DOCU,ref_docu);
      if dbstatus >= 90 then goto exit;
      assign (f,nom_fichier);
      rewrite (f);
      dbfpath (CO, DOCU, DOC);
      if dbstatus >= 90 then goto exit;
```

```
      {on écrit le fichier}
      while dbfound do
        begin
             for i := 1 to length (CO.LIGNEDOC) do
             write (f,CO.LIGNEDOC[i]);
             dbnpath (CO, DOCU, DOC);
             if dbstatus >= 90 then goto exit;
        end;
      close (f);

      dbcommit;
      code_retour := 0;

      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


procedure detruire_corps ;

{détruit le corps du document ou de sa référence stocké}
{dans la BD}

label exit ;

var  OBJ :TOBJET;
     DOCU : TDOCUMENT;
     COR : TCORPS;

begin
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     dbfpath (DOCU, OBJ, DOO);
     if dbstatus >= 90 then goto exit;

     dbfpath (COR, DOCU, DOC);
     if dbstatus >= 90 then goto exit;
     while dbfound do
       begin
            dbdelete (CORPS,COR);
            if dbstatus >= 90 then goto exit;
            dbfpath (COR,DOCU, DOC);
            if dbstatus >= 90 then goto exit;
       end;
     code_retour := 0;
     exit: ;
     if dbstatus > 10 then code_retour := dbstatus;
end;

END.
```

```
UNIT STRUCTUR;

INTERFACE

USES DOS, TYPBUR,DBMS,DECLAR ;

procedure creer_nouveau_type (nouv_type : NOUVEAUTYPE ;
                              var code_retour : integer);

procedure supprimer_type_objet (type_objet : TYPEOBJET ;
                              var code_retour : integer);
procedure creer_relation
                    (typeobjet_rg,typeobjet_rb : TYPEOBJET;
                              var code_retour :integer);

procedure supprimer_relation
                    (typeobjet_rg, typeobjet_rb : TYPEOBJET;
                              var code_retour : integer);

procedure modifier_attributs_type ( type_objet : TYPEOBJET;
                                    nouv_type : NOUVEAUTYPE;
                              var code_retour : integer);

procedure donner_premier_type (nouv_type : NOUVEAUTYPE;
                              var code_retour : integer);

procedure donner_type_suivant (nouv_type : NOUVEAUTYPE ;
                              var code_retour : integer);

procedure connaitre_caract_bureau
                           (fichier_structure : NOMFICHIER;
                              var code_retour :integer);

IMPLEMENTATION

procedure creer_nouveau_type  ;

label exit ;

var CAT : TCATEGORIEOBJET;

begin
      if (nouv_type.type_objet = 'BUREAU') or
         (nouv_type.type_objet = 'DOCUMENT')
       then
            begin
                 code_retour := 2;
                 dbstatus := 0;
                 goto exit;
            end;

      CAT.TYPEOBJETCAT := nouv_type.type_objet;
      CAT.NBOCCUPERMISES := nouv_type.nb_occu_permises;
      CAT.NBOCCUPRESENTES := nouv_type.nb_occu_presentes;
```

```
      CAT.DERNIERCODEATTRIBUT := nouv_type.dernier_code;
      dbcreate (CATEGORIEOBJET,CAT);
                  case dbstatus of
                                    0 : code_retour := 0;
                                    2 : code_retour := 2;
                                    else goto exit;
                    end;

      dbcommit;
      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


procedure supprimer_type_objet  ;

label exit ;

var CAT : TCATEGORIEOBJET;
    INCL : TINCLUSION;

begin
     if (type_objet = 'BUREAU') or (type_objet = 'DOCUMENT')
       then
            begin
                  code_retour := 40;
                  goto exit;
            end;

      CAT.TYPEOBJETCAT := type_objet ;
      dbid (CATEGORIEOBJET , CAT);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                        begin
                             code_retour := 41;
                             goto exit;
                        end;

      {enlever les inclusions où le type d''objet }
      {est de rangement}
      dbfpath (INCL,CAT,CB);
      if dbstatus>= 90 then goto exit;
      while dbfound do
                       begin
                            dbdelete (INCLUSION,INCL);
                            if dbstatus >= 90 then goto exit;
                            dbfpath (INCL,CAT,CB);
                            if dbstatus >= 90 then goto exit;
                        end;

      {enlever les inclusions où le type d''objet}
      {est rangeable }
      dbfpath (INCL,CAT,BC);
      if dbstatus>= 90 then goto exit;
```

```
      while dbfound do
                    begin
                         dbdelete (INCLUSION,INCL);
                         if dbstatus >= 90 then goto exit;
                         dbfpath (INCL,CAT,BC);
                         if dbstatus >= 90 then goto exit;
                       end;

      {on supprime le type d'objet}
      dbdelete (CATEGORIEOBJET, CAT);
      if dbstatus >= 90 then goto exit;

      dbcommit;
      code_retour := 0;
      exit : ;

     if dbstatus > 10 then code_retour := dbstatus;

end;



procedure creer_relation ;

label exit;

var CATRG {RG pour de rangement},
    CATRB {RB pour rangeable} : TCATEGORIEOBJET;
    INCL : TINCLUSION;

begin
     CATRG.TYPEOBJETCAT := typeobjet_rg;
     dbid (CATEGORIEOBJET,CATRG);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 4;
                            goto exit ;
                       end;

     CATRB.TYPEOBJETCAT := typeobjet_rb;
     dbid (CATEGORIEOBJET, CATRB);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                       begin
                           code_retour := 3;
                           goto exit ;
                       end;

if (typeobjet_rg = 'DOCUMENT') or (typeobjet_rb = 'BUREAU')
           then
               begin
                    code_retour := 40;
                    goto exit ;
               end;
```

```
    {on vérifie si l'inclusion existe déjà sinon on la crée}
    INCL.COUPLE := concat (typeobjet_rg,typeobjet_rb);
    dbid (INCLUSION, INCL);
    if dbstatus >= 90 then goto exit;
    if  dbfound then
                    begin
                         code_retour := 44;
                         goto exit ;
                    end
               else
                    begin
                         dbcreate (INCLUSION,INCL);
                         if dbstatus >= 80 then goto exit;
                         dbinsert ( INCL, CATRG, CB);
                         if dbstatus >= 80 then goto exit;
                         dbinsert ( INCL, CATRB ,BC);
                         if dbstatus >= 80 then goto exit;
                    end;

     dbcommit;
     code_retour := 0;
     exit : ;
     if dbstatus > 10 then code_retour := dbstatus;
  end;



procedure supprimer_relation ;

label exit ;

var CATRG {RG pour de rangement},
    CATRB {RB pour rangeable} : TCATEGORIEOBJET;
    INCL : TINCLUSION;

begin
      CATRG.TYPEOBJETCAT := typeobjet_rg;
      dbid (CATEGORIEOBJET,CATRG);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                          begin
                               code_retour := 4;
                               goto exit ;
                          end;

       CATRB.TYPEOBJETCAT := typeobjet_rb;
       dbid (CATEGORIEOBJET, CATRB);
       if dbstatus >= 90 then goto exit;
       if not dbfound then
                            begin
                                 code_retour := 3;
                                 goto exit ;
                             end;
```

```
      {on recherche si l'inclusion existe si oui on la }
      {supprime}
      INCL.COUPLE := concat (typeobjet_rg, typeobjet_rb);
      dbid (INCLUSION, INCL);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                        begin
                             code_retour := 45;
                             goto exit ;
                        end
                      else
                          begin
                               dbdelete (INCLUSION,INCL);
                               if dbstatus >= 90
                                  then goto exit;
                           end;

      dbcommit;
      code_retour := 0;
      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;
 end;


procedure modifier_attributs_type  ;

label exit;

var CAT : TCATEGORIEOBJET;

begin
     {on recherche l'objet}
     CAT.TYPEOBJETCAT := type_objet;
     dbid (CATEGORIEOBJET, CAT);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
        begin
             code_retour := 41;
             goto exit ;
        end
                    else
        begin
        {on affecte les nouvelles valeurs}
        CAT.NBOCCUPERMISES := nouv_type.nb_occu_permises;
        CAT.NBOCCUPRESENTES := nouv_type.nb_occu_presentes;
        CAT.DERNIERCODEATTRIBUT := nouv_type.dernier_code;

        {on répercute les modifications}
        dbmodify (CATEGORIEOBJET,CAT);
        if dbstatus >= 80 then goto exit;
        end;

     dbcommit;
     code_retour := 0;
```

```
     exit : ;
     if dbstatus > 10 then code_retour := dbstatus;
   end;

procedure donner_premier_type;

label exit;

var CAT : TCATEGORIEOBJET;

begin
     dbfirst (CATEGORIEOBJET,CAT);
     {on est sur d'avoir 2 types : 'BUREAU' et 'DOCUMENT'}
     if dbstatus >= 90 then goto exit;
     nouv_type.type_objet := CAT.TYPEOBJETCAT;
     nouv_type.nb_occu_permises := CAT.NBOCCUPERMISES;
     nouv_type.nb_occu_presentes := CAT.NBOCCUPRESENTES;
     nouv_type.dernier_code := CAT.DERNIERCODEATTRIBUT;
     code_retour := 0;
     exit: ;
     if dbstatus > 10 then code_retour := dbstatus;
end;

procedure donner_type_suivant;

label exit;

var CAT : TCATEGORIEOBJET;

begin
      CAT.TYPEOBJETCAT := nouv_type.type_objet;
      dbid (CATEGORIEOBJET, CAT);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                          begin
                               code_retour := 41;
                               goto exit;
                          end;
     nouv_type.type_objet := CAT.TYPEOBJETCAT;
     nouv_type.nb_occu_permises := CAT.NBOCCUPERMISES;
     nouv_type.nb_occu_presentes := CAT.NBOCCUPRESENTES;
     nouv_type.dernier_code := CAT.DERNIERCODEATTRIBUT;
     code_retour := 0;
     exit: ;
     if dbstatus > 10 then code_retour := dbstatus;
end;

procedure connaitre_caract_bureau ;

label exit;
type fich = TEXT;
var f : fich ;
    CAT : TCATEGORIEOBJET;
    INCL : TINCLUSION;
```

```
begin
     dbclear (CAT);
     assign (f, fichier_structure);
     rewrite (f);
     dbfirst (CATEGORIEOBJET, CAT);
     if dbstatus >= 90 then goto exit;
     while dbfound do
         begin
               writeln (f);
               writeln (f, 'type d''objet');
               writeln (f);
               writeln (f, CAT.TYPEOBJETCAT);
               writeln (f);
          writeln (f, 'nombre d''occurrences permises : ');
               writeln (f);
               writeln (f,CAT.NBOCCUPERMISES);
               writeln (f);
          writeln (f, 'nombre d''occurrences presentes :');
               writeln (f, CAT.NBOCCUPRESENTES);
               writeln (f, 'dernier code généré :');
               writeln (f, CAT.DERNIERCODEATTRIBUT);


               dbfpath (INCL, CAT,CB);
               if dbstatus >= 90 then goto exit;
               if dbfound then
                    begin
                         writeln (f);
                         writeln
                          (f, 'relations d''inclusion où  ',
                           CAT.TYPEOBJETCAT,
                           ' est objet de rangement');
                         while dbfound do
                           begin
                                writeln (f);
                                writeln (f, INCL.COUPLE);
                                dbnpath (INCL,CAT,CB);
                                if dbstatus >= 90
                                   then goto exit;
                            end;
                    end;
                dbfpath (INCL, CAT,BC);
                if dbstatus >= 90 then goto exit;
                if dbfound then
                     begin
                         writeln (f);
                         writeln
                           (f, 'relations d''inclusion où',
                            CAT.TYPEOBJETCAT,
                            ' est objet rangeable');
                         while dbfound do
                             begin
                                  writeln (f);
                                  writeln (f, INCL.COUPLE);
```

```
                                    dbnpath (INCL,CAT,BC);
                                    if dbstatus >= 90
                                       then goto exit;
                                end;
                            end;
                        dbnext (CATEGORIEOBJET , CAT);
                        if dbstatus >= 90 then goto exit;
                        writeln (f);
                        writeln (f);

                end;

            close (f);
            code_retour := 0;
            exit: ;
            if dbstatus >10 then code_retour := dbstatus;

        end;

    END.
```

```
UNIT ACCES;

INTERFACE

USES DOS,TYPBUR , DBMS ,DECLAR , OUTIL;

procedure donner_premier (type_objet : TYPEOBJET ;
                          var ref_objet : DBKEY;
                          var code_retour : integer);

procedure donner_suivant (type_objet : TYPEOBJET;
                          var ref_objet : DBKEY;
                          var code_retour : integer);

procedure acces_direct (nom_objet : NOMOBJET;
                          var ref_objet : DBKEY ;
                          var code_retour : integer);

procedure acces_premier (ref_contenant : DBKEY;
                          var ref_contenu : DBKEY;
                          var code_localisation :integer;
                          var code_retour : integer);

procedure acces_suivant (ref_contenant : DBKEY ;
                          var ref_contenu : DBKEY;
                          var code_localisation :integer;
                          var code_retour : integer);

IMPLEMENTATION

procedure donner_premier;

label exit;

var CAT : TCATEGORIEOBJET;
    OBJ : TOBJET;

begin
     {on recherche le type de l'objet}
     CAT.typeobjetcat := type_objet;
     dbid (CATEGORIEOBJET,CAT);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                    begin
                         code_retour := 41;
                         goto exit ;
                end;

     {on recherche la première occurrence du type d'objet}
     dbfpath (OBJ,CAT,CO);
     if dbstatus >= 90 then goto exit;
```

```
      if not dbfound
                    then
                         begin
                              {s'il n'existe pas d'occurrence}
                              {ref_objet est mis à null}
                              code_retour := 1;
                              dbclear (OBJ);
                              ref_objet := OBJ.xxref;
                              goto exit;
                         end
                     else
                         begin
                              ref_objet := OBJ.XXref;
                              code_retour := 0;
                         end;

       exit : ;
       if dbstatus > 10 then code_retour := dbstatus;
    end;

procedure donner_suivant ;

label exit ;

var CAT : TCATEGORIEOBJET;
    OBJ ,CONTA : TOBJET;

begin
     CAT.typeobjetcat := type_objet;
     dbid (CATEGORIEOBJET,CAT);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 41;
                            goto exit ;
                       end;
      OBJ.xxref := ref_objet;
      if dbnull (OBJ) {alors le comportement doit être le}
                      {même que pour donner_premier}
            then
               begin
                    donner_premier (type_objet, ref_objet,
                                              code_retour);
                    dbstatus := 0;
                    goto exit;
                end;

       {on recherche l'occurrence précédente càd celle}
       {fournie en entrée}
       dbdirect (OBJET,OBJ, ref_objet);
       if dbstatus >= 90 then goto exit;
       if dbstatus = 30 then goto exit;
```

```
        if not dbfound then
                          begin
                               code_retour := 3;
                               goto exit;
                          end;
        if OBJ.TYPEOBJET <> type_objet
           then
               begin
                    code_retour := 5;
                    goto exit;
               end;

        {on recherche l'occurrence suivante}
        dbnpath (OBJ,CAT,CO);
        if dbstatus >= 90 then goto exit;
        if not dbfound
                 then
                     begin
                          {s'il n'y a pas d'occurrence}
                          {suivante alors ref_objet est}
                          {mis à null}
                           code_retour := 1;
                           dbclear (OBJ);
                           ref_objet := OBJ.xxref;
                           goto exit;
                      end
                 else
                     begin
                          ref_objet := OBJ.xxref;
                          code_retour := 0;
                     end;

          exit: ;
          if dbstatus > 10 then
                               begin
                                    code_retour := dbstatus;
                                    dbclear (OBJ);
                                    ref_objet := OBJ.xxref;
                               end;
  end;


procedure acces_direct  ;

label exit;

var OBJ : TOBJET;

begin
      OBJ.NOMOBJET := nom_objet;
      dbid  (OBJET, OBJ);
      If dbstatus >= 90 then goto exit;
```

```
      if not dbfound then begin
                                code_retour := 1;
                                dbclear (OBJ);
                                ref_objet := OBJ.xxref;
                                goto exit;
                           end
                      else begin
                                code_retour := 0;
                                ref_objet := OBJ.xxref;
                            end;
       exit : ;
       if dbstatus > 10 then code_retour := dbstatus ;
 end;

procedure acces_premier ;

label exit;

var CONTENU_ORIGINE, CONTENU_REEL : TOBJET;
    CONTENANT, CONTENU : TOBJET;
    loc_origine, loc_reel : boolean;
    localisation : integer;
    val_string_reel, val_string_origine : string [30];
    val_nombre_reel, val_nombre_origine : real;
    an, mo, jo : real;

begin
     {on recherche l'objet dont on veut le premier contenu}
     dbdirect (OBJET, CONTENANT,ref_contenant);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then
                          begin
                          dbstatus := 31 ;
                          goto exit;
                       end;
     if not dbfound then
                        begin
                             code_retour := 4 ;
                             goto exit ;
                         end;
     if CONTENANT.TYPEOBJET = 'DOCUMENT' then
        begin
             code_retour := 40;
             goto exit;
        end;

     {on recherche le premier contenu dont le contenant}
     {est lieu d'origine}
     dbfpath (CONTENU_ORIGINE, CONTENANT,LOD);
     if dbstatus >= 90 then goto exit;
     if dbfound then
                    begin
                          loc_origine := true;
                    end
                else
```

```
                begin
                      loc_origine := false;
                end;

{on recherche le premier contenu dont le contenant}
{est le lieu réel}
dbfpath (CONTENU_REEL, CONTENANT,LO);
if dbstatus >= 90 then goto exit;
if dbfound then
                begin
                      loc_reel := true;
                end
           else
                begin
                      loc_reel := false;
                end;

{on teste s'il n'y a pas de contenu}
if (loc_reel = false) and (loc_origine = false) then
   begin
        code_retour := 1;
        dbclear (CONTENU);
        dbcopyref (CONTENU.xxref, ref_contenu);
        goto exit ;
   end;

{s'il n'y apas de contenu réel...}
if loc_reel = false then
   begin
        ref_contenu := CONTENU_ORIGINE.xxref;
        code_retour := 0;
        code_localisation := 2;
        goto exit;
   end;

{s'il n'y a pas de contenu origine...}
if loc_origine = false then
   begin
        ref_contenu := CONTENU_REEL.xxref;
        code_retour := 0;
        code_localisation := 1;
        goto exit;
   end;

{on teste si le contneu a sa place réelle et d'origine}
{confondues}
if dbequal (CONTENU_REEL.xxref, CONTENU_ORIGINE.xxref)
 then
   begin
        ref_contenu := CONTENU_ORIGINE.xxref;
        code_retour := 0;
        code_localisation := 0;
        goto exit;
   end
```

```
else   {dans le cas contraire il faut prendre parmi}
       {les contenus réels et d'origine celui qui}
       {dans le classement du contenant vient en tete}
  begin
  if CONTENANT.ATTRIBUTCLASSEMENTOB in [1,2,4,6]
     then
     begin
     case CONTENANT.ATTRIBUTCLASSEMENTOB of
     1 : begin
         val_string_reel := CONTENU_REEL.TYPEOBJET;
         val_string_origine := CONTENU_ORIGINE.TYPEOBJET;
         end;
     2 : begin
         val_string_reel := CONTENU_REEL.NOMOBJET;
         val_string_origine := CONTENU_ORIGINE.NOMOBJET;
         end;
     4 : begin
         val_string_reel :=
                   CONTENU_REEL.CREATEURRESPONSABLE;
         val_string_origine :=
                   CONTENU_ORIGINE.CREATEURRESPONSABLE;
         end;
     6 :begin
        val_string_reel := CONTENU_REEL.MOTCLEOBJET1;
        val_string_origine :=
                   CONTENU_ORIGINE.MOTCLEOBJET1;
        end;
     end;
     end;
  if CONTENANT.ATTRIBUTCLASSEMENTOB in [3,5,7]
     then
     begin
     case CONTENANT.ATTRIBUTCLASSEMENTOB of
      3 : begin
an := CONTENU_REEL.DATECREATIONOBJET.ANNEE * 10000;
mo := CONTENU_REEL.DATECREATIONOBJET.MOIS * 1000 ;
jo := CONTENU_REEL.DATECREATIONOBJET.JOUR ;
          val_nombre_reel  := an + mo + jo ;
          Str (val_nombre_reel, val_string_reel);

an := CONTENU_ORIGINE.DATECREATIONOBJET.ANNEE * 10000;
mo := CONTENU_ORIGINE.DATECREATIONOBJET.MOIS * 1000 ;
jo := CONTENU_ORIGINE.DATECREATIONOBJET.JOUR ;
          val_nombre_origine  := an + mo + jo ;
          Str (val_nombre_origine, val_string_origine);
          end;
      5 : begin
an := CONTENU_REEL.DATEDERMAJOBJET.ANNEEMAJ * 10000;
mo := CONTENU_REEL.DATEDERMAJOBJET.MOISMAJ * 100 ;
jo := CONTENU_REEL.DATEDERMAJOBJET.JOURMAJ ;
          val_nombre_reel  := an + mo + jo ;
          Str (val_nombre_reel, val_string_reel);

an := CONTENU_ORIGINE.DATEDERMAJOBJET.ANNEEMAJ * 10000;
mo := CONTENU_ORIGINE.DATEDERMAJOBJET.MOISMAJ * 1000 ;
```

```
        jo := CONTENU_ORIGINE.DATEDERMAJOBJET.JOURMAJ ;
              val_nombre_origine  := an + mo + jo ;
              Str (val_nombre_origine, val_string_origine);
              end;
           7 : begin
              val_nombre_reel := CONTENU_REEL.MOTCLEOBJET2;
              Str (val_nombre_reel, val_string_reel);
         val_nombre_origine := CONTENU_ORIGINE.MOTCLEOBJET2;
              Str (val_nombre_origine, val_string_origine);
              end;
           end;
         end;
        if val_string_origine < val_string_reel
           then
              if CONTENANT.MODECLASSEMENTOBJET = 1
                 then
                 ref_contenu :=  CONTENU_ORIGINE.xxref
                 else
                 ref_contenu := CONTENU_REEL.xxref
           else
                if CONTENANT.MODECLASSEMENTOBJET = 2
                 then
                 ref_contenu := CONTENU_REEL.xxref
                 else
                 ref_contenu :=  CONTENU_ORIGINE.xxref;
        if val_string_origine >=  val_string_reel
           then
              if CONTENANT.MODECLASSEMENTOBJET = 2
                 then
                 ref_contenu :=  CONTENU_ORIGINE.xxref
                 else
                 ref_contenu := CONTENU_REEL.xxref
           else
                 if CONTENANT.MODECLASSEMENTOBJET = 1
                    then
                    ref_contenu := CONTENU_REEL.xxref
                    else
                    ref_contenu :=  CONTENU_ORIGINE.xxref;

        {on calcule la localisation (réel ou origine )
        {ou les deux) du contenu}
        code_localisation := 0;
        calcul_localisation
                (ref_contenu,code_localisation,code_retour);
        if code_retour > 0 then goto exit;
        end;

     code_retour := 0;
     exit : ;
     if dbstatus > 10 then code_retour := dbstatus;

end;
```

```
procedure acces_suivant ;

label exit ;

var tete_reel, tete_origine, tete_res : cellule;
    p,q,r,queue_reel, queue_origine, queue_res : cellule;
    CONTENANT, CONTENU, CONT, CONTA,OBJ : TOBJET;
    an, mo, jo : real;
    string_ok : boolean;
    val_string : TYPSTR;
    val_nombre : real;
    loc_origine , loc_reel : boolean;
    val_localisation : integer;
    val_code_retour : integer;
    val_ref_contenu : DBKEY;
    localisation : integer;
    lieu : char;

begin

     dbdirect (OBJET, CONTENANT, ref_contenant);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then
                          begin
                               dbstatus := 31;
                               goto exit;
                          end;
     if not dbfound       {verification du contenant }
       then
          begin
               code_retour := 4 ;
               goto exit ;
          end;

     if CONTENANT.TYPEOBJET = 'DOCUMENT' then
        begin
             code_retour := 40;
             goto exit;
        end;
     { cas où ref_contenu = null d'où le}
     { comportement = acces_premier :}

     dbdirect (OBJET, CONTENU,ref_contenu);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 3;
                            goto exit;
                       end;
     if dbnull (CONTENU)  {alors le comportement doit }
                          {être identique à acces_premier}
        then
```

```
       begin
            acces_premier (ref_contenant, ref_contenu,
                       code_localisation, code_retour);
            dbstatus := 0;
            goto exit ;
       end;
{ on controle si ref_contenu recu reference bien}
{un objet du contenant   si on a testpath ...}

dbdirect (OBJET, CONTENU, ref_contenu);
if dbstatus >= 90 then goto exit;
if dbstatus = 30 then goto exit;
if not dbfound then
                  begin
                       dbstatus := 0;
                       code_retour := 5;
                       goto exit;
                  end;
dbfpath (CONTA, CONTENU, -LO);
if dbstatus >= 90 then goto exit;
if not dbfound then
  begin
     dbfpath (CONTA, CONTENU, -LOD);
     if dbstatus >= 90 then goto exit;
     if not dbequal (CONTA.xxref,CONTENANT.xxref)
                  then
                      begin
                            code_retour := 5;
                            goto exit;
                      end;

  end;
{ on procede à la lecture des contenus reels et on}
{ met ceux-ci dans une liste chainee qui est triée}
{ (les contenus présents sont déjà triés)}

lieu := 'r';
lecture_contenus
    (CONTENANT.xxref, lieu, tete_reel,code_retour);
if code_retour > 0 then goto exit;

{ on procede à la lecture des contenus origine et on }
{met ceux-ci dans une liste chainee qui est triée}
{(les contenus présents sont déjà triés)}

lieu := 'o';
lecture_contenus
   (CONTENANT.xxref, lieu, tete_origine, code_retour);
if code_retour > 0 then goto exit;

if tete_reel = nil then
   tete_res := tete_origine;
if tete_origine = nil then
   tete_res := tete_reel;
```

```
{si le mode de classement du contenant est non trié }
{ on concatène les 2 listes}
if (CONTENANT.MODECLASSEMENTOBJET = 0 ) and
     ( (tete_reel <> nil) and (tete_origine <> nil))
      then
          begin
               {on concatene les deux listes }
               tete_res := tete_origine;
               p := tete_origine;
               while p <> nil do
                  begin
                        new (r);
                        r^.chaine  := p^.chaine;
                        r^.ref_objet := p^.ref_objet;
                        if tete_res = nil
                            then tete_res := r
                            else queue_res^.next := r;
                        queue_res := r;
                        p := p^.next;
                  end;
               p := tete_reel ;
               while p <> nil do
                begin
                      new (r);
                      r^. chaine := p^.chaine;
                      r^.ref_objet := p^.ref_objet;
                      queue_res^.next := r;
                      queue_res := r;
                      p := p^.next;
                end;

               end;

           { fusion des deux listes }
if CONTENANT.MODECLASSEMENTOBJET <> 0 then

 begin

  tete_res := nil;
  queue_res := nil;
  p := tete_origine;
  q := tete_reel;

  while ( p <> nil) or (q <> nil ) do
  {si les 2 listes sont vides,la liste}
  { résultante est vide}

    begin
      if p = nil then
         begin
               new (r);
               r^.chaine := q^.chaine;
               r^.ref_objet := q^.ref_objet;
               r^.next := nil;
               if tete_res = nil
```

```
            then tete_res := r
            else queue_res^.next := r;
         queue_res := r;
         q := q^.next;
   end;

if q = nil then
   begin
         new (r);
         r^.chaine := p^.chaine;
         r^.ref_objet := p^.ref_objet;
         r^.next := nil;
         if tete_res = nil
            then tete_res := r
            else queue_res^.next := r;
         queue_res := r;
         p := p^.next;
    end;


if p^.chaine < q^.chaine
 then
    begin
         new (r);
         if CONTENANT.MODECLASSEMENTOBJET = 1
            then
                begin
                      r^.chaine := p^.chaine;
                      r^.ref_objet := p^.ref_objet;
                      p := p^.next
                end
             else
                 begin
                      r^.chaine := q^.chaine;
                      r^.ref_objet := q^.ref_objet;
                      q := q^.next
                 end;
         r^.next := nil;
         if tete_res = nil
            then tete_res := r
            else queue_res^.next := r;
         queue_res := r;
    end
else
    begin
         if p^.chaine = q^.chaine
            then
                begin
                {il regarder si ce n'est pas le}
                { même objet !!!}
                if dbequal
                     (p^.ref_objet,q^.ref_objet)
                   then
                       begin
                            new (r);
```

```
                                r^.chaine := q^.chaine;
                              r^.ref_objet := q^.ref_objet;
                                        q := q^.next;
                                        p := p^.next;
                                   r^.next := nil;
                                   if tete_res = nil
                                      then tete_res := r
                                       else
                                      queue_res^.next := r;

                                      queue_res := r;
                                  end
                               else
                                   begin
                                        new (r);
                                  r^.chaine := p^.chaine;
                               r^.ref_objet := p^.ref_objet;
                                        p := p^.next;
                                    r^.next := nil;
                                 if tete_res = nil
                                    then tete_res := r
                                    else queue_res^.next := r;
                                      queue_res := r;
                                    end;
                            end
                         else
                             begin
                                  new (r);
                               if CONTENANT.MODECLASSEMENTOBJET = 1
                               then
                                begin
                                    r^.chaine := q^.chaine;
                                    r^.ref_objet := q^.ref_objet; q
                                    := q^.next
                                 end
                                else
                                 begin
                                     r^.chaine := p^.chaine;
                                     r^.ref_objet := p^.ref_objet;
                                     p := p^.next
                                 end;
                               r^.next := nil;
                               if tete_res = nil
                                  then tete_res := r
                                  else queue_res^.next := r;
                               queue_res := r;
                               end;
          end;
     end;
end;
     { parcours de la liste résultante et recherche}
     { de l'ancienne valeur de ref_contenu}

     { on est sur que ref_contenu ancien est dans la}
     {liste car on a teste precedemment }
```

```
r := tete_res ;
while r <> nil do
begin
  if dbequal (r^.ref_objet , ref_contenu)
     then
    {on a trouvé l'ancienne valeur de ref_contenu}
        begin
              r := r^.next;
              if r = nil
                  then
              {il n'y a pas de suivant}
                      begin
                           code_retour := 1;
                           goto exit;
                      end
                   else
      {la nouvelle valeur de ref_contenu est la }
      {valeur suivante}
                        begin
                      ref_contenu := r^.ref_objet;
                        end;
                end;
  r := r^.next;

end;

{ calcul du code de localisation : }
calcul_localisation
    (ref_contenu,code_localisation,code_retour);
if code_retour > 0 then goto exit;

code_retour := 0;
exit : ;
if dbstatus > 10 then code_retour := dbstatus;

end;

END.
```

```
UNIT CREAMAJ;

INTERFACE

USES DOS, TYPBUR, DBMS, DECLAR, OUTIL;

procedure creer_objet (var objet_a_creer : OBJET_ATTR ;
                  var ref_contenant ,ref_contenu : DBKEY;
                              var code_retour :integer);

procedure detruire_objet (var ref_objet : DBKEY ;
                              var code_retour : integer);

procedure caracteristiques (ref_objet : DBKEY ;
                              var objet_car : OBJET_ATTR;
                              var code_retour : integer);

procedure modifier_attributs (ref_objet : DBKEY ;
                              var obj_mod : OBJET_ATTR;
                              var code_retour : integer);

procedure localisation_origine (ref_contenu : DBKEY;
                              var ref_contenant : DBKEY;
                              var code_retour : integer);

procedure localisation_reelle (ref_contenu : DBKEY;
                              var ref_contenant : DBKEY;
                              var code_retour : integer);

procedure changer_lieu_origine
                        (ref_contenu,ref_contenant : DBKEY;
                              var code_retour : integer);

procedure deplacer_objet (ref_contenu,ref_contenant : DBKEY;
                              var code_retour : integer);

procedure reinserer_place_origine  (ref_objet : DBKEY ;
                              var code_retour : integer);


IMPLEMENTATION

procedure creer_objet ;

label exit;

var OBJ_CREER, CONTENANT : TOBJET;
    DOCU_CREER : TDOCUMENT ;
    CAT : TCATEGORIEOBJET;
    INCL : TINCLUSION;
    f : file of char;
    id : string [3];
    lieu : char;
    code : string [27];
```

```
begin
     {on teste si l'objet à créer existe bien}
     CAT.TYPEOBJETCAT := objet_a_creer.typeobjet;
     dbid (CATEGORIEOBJET, CAT);
     if dbstatus >= 90 then goto exit;
     if not dbfound
         then
              begin
                   code_retour := 41;
                   goto exit;
              end;

     if objet_a_creer.typeobjet = 'BUREAU'
                     then
                         begin
                               code_retour := 40;
                               goto exit ;
                         end;

     {on teste si on ne dépasse pas le nombre d'occurrences}
     {permises}
     if CAT.NBOCCUPERMISES < 0
       then
       else
           begin
            CAT.NBOCCUPRESENTES := CAT.NBOCCUPRESENTES + 1 ;
            if CAT.NBOCCUPRESENTES > CAT.NBOCCUPERMISES
                 then
                     begin
                          code_retour := 43;
                          goto exit;
                     end;
           end;

     { on teste si le contenant existe }
     dbdirect (OBJET, CONTENANT, ref_contenant);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then
                          begin
                                dbstatus := 31;
                                goto exit;
                          end;
     if not dbfound
         then
              begin
                   code_retour := 4 ;
                   goto exit;
              end;

     {on teste si l'inclusion est permise dans l'ER}
     INCL.COUPLE :=
      concat (CONTENANT.TYPEOBJET, objet_a_creer.typeobjet);
     dbid (INCLUSION,INCL);
     if dbstatus >= 90 then goto exit;
```

```
if not dbfound
    then
         begin
              code_retour := 42;
              goto exit;
         end;

{on teste si le fichier (le corps ou la référence)}
{est disponible}
if objet_a_creer.typeobjet = 'DOCUMENT' then
    begin
          assign (f, objet_a_creer.nomfichier);
          {$I-} reset (f) {$I+};
          if IOresult = 1 then
                              begin
                                   code_retour := 50;
                                   goto exit;
                              end;
    end;

{on teste si l'utilisateur veut donner un code ou un}
{nom à l'objet}
if objet_a_creer.nomobjet = ''
 then
    begin
          CAT.DERNIERCODEATTRIBUT :=
                         CAT.DERNIERCODEATTRIBUT + 1;
          id := copy (objet_a_creer.typeobjet, 1,3);
          Str (CAT.DERNIERCODEATTRIBUT,code);
          objet_a_creer.nomobjet :=
                               Concat (id,'/',code);
    end
 else
      begin
           id := copy (objet_a_creer.typeobjet, 1,3);
           objet_a_creer.nomobjet :=
               Concat (id,'/',objet_a_creer.nomobjet);
        end;

 {on crée l'occurrence de l'objet}
 OBJ_CREER.TYPEOBJET := objet_a_creer.typeobjet;
 OBJ_CREER.NOMOBJET := objet_a_creer.nomobjet;
 date (annee, mois , jour);
 OBJ_CREER.DATECREATIONOBJET.JOUR := jour;
 OBJ_CREER.DATECREATIONOBJET.MOIS := mois;
 OBJ_CREER.DATECREATIONOBJET.ANNEE := annee;
 OBJ_CREER.DATEDERMAJOBJET.JOURMAJ := jour;
 OBJ_CREER.DATEDERMAJOBJET.MOISMAJ := mois;
 OBJ_CREER.DATEDERMAJOBJET.ANNEEMAJ := annee;
 OBJ_CREER.CREATEURRESPONSABLE :=
            objet_a_creer.createurresponsable;
 OBJ_CREER.MODECLASSEMENTOBJET :=
            objet_a_creer.modeclassementobjet;
 OBJ_CREER.ATTRIBUTCLASSEMENTOB :=
            objet_a_creer.attributclassementob;
```

```
 OBJ_CREER.MOTCLEOBJET1 := objet_a_creer.motcleobjet1;
 OBJ_CREER.MOTCLEOBJET2 := objet_a_creer.motcleobjet2;
 OBJ_CREER.CONFIDENTIALITE :=
           objet_a_creer.confidentialite ;
 OBJ_CREER.MOTDEPASSE := objet_a_creer.motdepasse;

dbcreate (OBJET, OBJ_CREER);
if dbstatus >= 90 then goto exit;
if dbstatus = 2
    then
         begin
              code_retour := 2;
              dbstatus := 0;
              goto exit;
         end;

{si l'objet est un document : }
if objet_a_creer.typeobjet = 'DOCUMENT' then
   begin
    DOCU_CREER.MODEREPRESENTATION :=
                objet_a_creer.moderepresentation;
    DOCU_CREER.TYPEINFO := objet_a_creer.typeinfo;
    DOCU_CREER.REFERENCEORIGINAL :=
                objet_a_creer.referenceoriginal;
    DOCU_CREER.NOMFICHIER := objet_a_creer.nomfichier;
    DOCU_CREER.STOCKAGE := objet_a_creer.stockage ;
    DOCU_CREER.COPIEDE := objet_a_creer.copiede;
    DOCU_CREER.DATECOPIE.JOURCOPIE := 0;
    DOCU_CREER.DATECOPIE.MOISCOPIE := 0;
    DOCU_CREER.DATECOPIE.ANNEECOPIE := 0;
    DOCU_CREER.ELECTRONIQUE :=
                objet_a_creer.electronique;

    dbcreate (DOCUMENT, DOCU_CREER);
    if dbstatus >= 80 then goto exit;

    {on relie document à objet : }
    dbinsert (DOCU_CREER, OBJ_CREER , DOO);
    if dbstatus >= 80 then goto exit;

    {on crée le corps du document et on le rattrache}
    {à document}
    decoupe_fichier (DOCU_CREER.xxref,
              objet_a_creer.nomfichier, code_retour);
    if code_retour > 0 then goto exit;
   end;

{une fois l'objet créé on réprecute les modifications}
{dans l'ER}
dbmodify (CATEGORIEOBJET, CAT);
if dbstatus >= 80 then goto exit;

{on relie l'occurrence d'objet à la structure de l'ER}
dbinsert (OBJ_CREER, CAT, CO);
if dbstatus >= 80 then goto exit;
```

```
      {on classe l'objet créé dans son contenant qui est à}
      {la fois lieu d'origine et lieu réel}

      lieu := 'r';  {r pour réel}
      ref_contenu := OBJ_CREER.xxref;
      inserer_nouveau_element (ref_contenu,lieu,
                               ref_contenant,code_retour);
      if code_retour > 0 then goto exit;

      lieu := 'o';   {o pour origine}
      inserer_nouveau_element (OBJ_CREER.xxref,lieu,
                               ref_contenant,code_retour);
      if code_retour > 0 then goto exit;

     {on sauve dans la BD}
      dbcommit;

      code_retour := 0;

      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;

 end;


procedure detruire_objet ;

label exit ;

var CONTENANT,CONTENU, OBJ : TOBJET;
    COR : TCORPS;
    DOCU : TDOCUMENT;
    TEXT : TTEXTELIBRE;
    CAT : TCATEGORIEOBJET;

begin
     dbdirect (OBJET, OBJ, ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound
         then
             begin
                   code_retour := 3 ;
                   goto exit ;
             end;
     if OBJ.TYPEOBJET = 'BUREAU'
                      then
                          begin
                                code_retour := 40;
                                goto exit ;
                          end;
```

```
if OBJ.TYPEOBJET = 'DOCUMENT'
   { dbtestpath (DOCU, OBJ, DOO)}
   then
       begin
            {on détruit d''abord le corps du document}
            dbfpath (DOCU, OBJ, DOO);
            if dbstatus >= 90 then goto exit;
            dbfpath (COR, DOCU, DOC);
            if dbstatus >= 90 then goto exit;
            while dbfound do
                  begin
                       dbdelete (CORPS, COR);
                       if dbstatus >= 90 then goto exit;
                       dbfpath (COR,DOCU,DOC);
                       if dbstatus >= 90 then goto exit;
                   end;

             {on détruit ce qui particularise le document}
             dbdelete (DOCUMENT, DOCU);
             if dbstatus >= 90 then goto exit;
       end;

{ vérifier si l'objet est vide}
{ s'il ne l'est pas on abandonne }

{on vérifie d'abord s'il contient des objets qui sont à}
{ leur place d''origine}
dbfpath (CONTENU, OBJ, LO);
if dbstatus >= 90 then goto exit;
if dbfound then
               begin
                     code_retour := 40;
                     goto exit;
               end;
{on vérifie s'il contient des objets qui sont}
{à leur place réelle}
dbfpath (CONTENU, OBJ, LOD);
if dbstatus >= 90 then goto exit;
if dbfound then
               begin
                     code_retour := 40;
                     goto exit;
               end;

{on détruit le texte libre associé à l'objet}
dbfpath (TEXT, OBJ, TXT);
if dbstatus >= 90 then goto exit;
if dbfound then
      { if dbtestpath (TEXTLIBRE, OBJ, TXT)}
   begin
         dbfpath (TEXT, OBJ, TXT);
         if dbstatus >= 90 then goto exit;
```

```
            while dbfound do
                  begin
                       dbdelete (TEXTELIBRE, TEXT);
                       if dbstatus >= 90 then goto exit;
                       dbfpath (TEXT, OBJ, TXT);
                       if dbstatus >= 90 then goto exit;
                  end;

         end;

      {on met à jour le compteur d'occurrences }
      {du type de l''objet}
      CAT.TYPEOBJETCAT := OBJ.TYPEOBJET;
      dbid (CATEGORIEOBJET,CAT);
      if dbstatus >= 90 then goto exit;
      CAT.NBOCCUPRESENTES := CAT.NBOCCUPRESENTES - 1;
      dbmodify (CATEGORIEOBJET, CAT);
      if dbstatus >= 80 then goto exit;

      {on met à jour la date de dernière mise à jour}
      {du contenant}
      dbfpath (CONTENANT, OBJ , - LOD);
      if dbstatus >= 90 then goto exit;
      maj_date_maj_contenant (CONTENANT.xxref, code_retour);
      if code_retour > 0  then goto exit;

      {on détruit l'occurrence d''objet}
      dbdelete (OBJET, OBJ);
      if dbstatus >= 90 then goto exit;

      { on sauve les modifications de la bd}
      dbcommit;
      code_retour := 0;

      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;

 end;


procedure caracteristiques ;

label exit;
var OBJ : TOBJET;
    DOCU : TDOCUMENT;

begin
     dbdirect (OBJET, OBJ, ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 3 ;
                            goto exit;
                       end;
```

```
objet_car.typeobjet := OBJ.TYPEOBJET ;
objet_car.nomobjet := OBJ.NOMOBJET ;
objet_car.datecreationobjet.jour :=
                        OBJ.DATECREATIONOBJET.JOUR ;
objet_car.datecreationobjet.mois :=
                        OBJ.DATECREATIONOBJET.MOIS ;
objet_car.datecreationobjet.annee :=
                        OBJ.DATECREATIONOBJET.ANNEE ;
objet_car.datedermaj.jourmaj :=
                        OBJ.DATEDERMAJOBJET.JOURMAJ ;
objet_car.datedermaj.moismaj :=
                        OBJ.DATEDERMAJOBJET.MOISMAJ ;
objet_car.datedermaj.anneemaj :=
                        OBJ.DATEDERMAJOBJET.ANNEEMAJ ;
objet_car.createurresponsable :=
                        OBJ.CREATEURRESPONSABLE;
objet_car.modeclassementobjet :=
                        OBJ.MODECLASSEMENTOBJET;
objet_car.attributclassementob :=
                        OBJ.ATTRIBUTCLASSEMENTOB;
objet_car.motcleobjet1 := OBJ.MOTCLEOBJET1;
objet_car.motcleobjet2 := OBJ.MOTCLEOBJET2;
objet_car.confidentialite := OBJ.CONFIDENTIALITE;
objet_car.motdepasse := OBJ.MOTDEPASSE;


if OBJ.TYPEOBJET = 'DOCUMENT' then
  begin
    dbfpath (DOCU, OBJ ,DOO);
    if dbstatus >= 90 then goto exit;
    objet_car.moderepresentation :=
                DOCU.MODEREPRESENTATION ;
    objet_car.typeinfo := DOCU.TYPEINFO ;
    objet_car.referenceoriginal :=
                DOCU.REFERENCEORIGINAL;
    objet_car.nomfichier:= DOCU.NOMFICHIER;
    objet_car.stockage := DOCU.STOCKAGE;
    objet_car.copiede := DOCU.COPIEDE;
    objet_car.datecopie.jourcopie :=
                DOCU.DATECOPIE.JOURCOPIE ;
    objet_car.datecopie.moiscopie :=
                DOCU.DATECOPIE.MOISCOPIE;
    objet_car.datecopie.anneecopie :=
                DOCU.DATECOPIE.ANNEECOPIE;
    objet_car.electronique := DOCU.ELECTRONIQUE;

  end
                  else
  begin
    objet_car.moderepresentation := '' ;
    objet_car.typeinfo := '' ;
    objet_car.referenceoriginal := '';
    objet_car.nomfichier:= '';
    objet_car.stockage := FALSE;
```

```
            objet_car.copiede := '';
            objet_car.datecopie.jourcopie :=  0 ;
            objet_car.datecopie.moiscopie := 0 ;
            objet_car.datecopie.anneecopie := 0;
            objet_car.electronique := FALSE;

        end ;
    code_retour := 0;

    exit: ;
    if dbstatus > 10 then
            begin
                code_retour := dbstatus;
                objet_car.typeobjet := '' ;
                objet_car.nomobjet := '' ;
                objet_car.datecreationobjet.jour := 0;
                objet_car.datecreationobjet.mois := 0 ;
                objet_car.datecreationobjet.annee :=  0;
                objet_car.datedermaj.jourmaj := 0 ;
                objet_car.datedermaj.moismaj := 0 ;
                objet_car.datedermaj.anneemaj := 0 ;
                objet_car.createurresponsable := '';
                objet_car.modeclassementobjet := 0;
                objet_car.attributclassementob := 0;
                objet_car.motcleobjet1 := '';
                objet_car.motcleobjet2 := 0;
                objet_car.confidentialite := false ;
                objet_car.motdepasse := '';
                objet_car.moderepresentation := '' ;
                objet_car.typeinfo := '' ;
                objet_car.referenceoriginal := '';
                objet_car.nomfichier:= '';
                objet_car.stockage := false;
                objet_car.copiede := '';
                objet_car.datecopie.jourcopie :=  0 ;
                objet_car.datecopie.moiscopie := 0 ;
                objet_car.datecopie.anneecopie := 0;
                objet_car.electronique := false;
            end;
end;


procedure modifier_attributs ;

label exit;
var OBJ, CONTENANTREEL, CONTENANTORIGINE : TOBJET;
    DOCU : TDOCUMENT;
    indicateur : integer;
    retrier : boolean;
    ens_attri_clst : set of 1..10;
    ancien_mode , nouveau_mode : integer;

begin
     dbdirect (OBJET, OBJ, ref_objet);
     if dbstatus >= 90 then goto exit;
```

```
     if dbstatus = 30 then goto exit;
     if not dbfound then
                        begin
                              code_retour := 3;
                              goto exit;
                        end;

     {ens_attri_clst permet de répertorier les valueurs }
     {d'attributs qui ont été modifiées, attributs qui}
     {peuvent être à la base du classement du contenant}
     ens_attri_clst := [];


{ on laisse le code au cas ou les dates seraient modifiables
if obj_mod.datecreationobjet.jour <> OBJ.DATECREATIONOBJET.JOUR
 then
OBJ.DATECREATIONOBJET.JOUR := obj_mod.datecreationobjet.jour;

if obj_mod.datecreationobjet.mois <> OBJ.DATECREATIONOBJET.MOIS
 then
OBJ.DATECREATIONOBJET.MOIS:= obj_mod.datecreationobjet.mois;

If obj_mod.datecreationobjet.annee <>
 OBJ.DATECREATIONOBJET.ANNEE then
OBJ.DATECREATIONOBJET.ANNEE:= obj_mod.datecreationobjet.annee;

if obj_mod.datedermaj.jourmaj <> OBJ.DATEDERMAJOBJET.JOURMAJ
 then
OBJ.DATEDERMAJOBJET.JOURMAJ := obj_mod.datedermaj.jourmaj;

If obj_mod.datedermaj.moismaj <> OBJ.DATEDERMAJOBJET.MOISMAJ
 then
OBJ.DATEDERMAJOBJET.MOISMAJ:= obj_mod.datedermaj.moismaj;

if obj_mod.datedermaj.anneemaj <> OBJ.DATEDERMAJOBJET.ANNEEMAJ
 then
OBJ.DATEDERMAJOBJET.ANNEEMAJ := obj_mod.datedermaj.anneemaj;
                    }

if obj_mod.createurresponsable <> OBJ.CREATEURRESPONSABLE
 then
     begin
     OBJ.CREATEURRESPONSABLE :=obj_mod.createurresponsable;
     ens_attri_clst := ens_attri_clst + [4];
     end;

     retrier := false;
     {permet de mémoriser s'il faut retrier suivant}
     {le nouvel attribut base de classement }
if obj_mod.attributclassementob <> OBJ.ATTRIBUTCLASSEMENTOB
 then
     begin
     OBJ.ATTRIBUTCLASSEMENTOB := obj_mod.attributclassementob;
     retrier := true;
```

```
      if obj_mod.attributclassementob = 0
          then   retrier := false;
      end;

if obj_mod.modeclassementobjet <> OBJ.MODECLASSEMENTOBJET
 then
     begin
          {nouveau mode et ancien mode servent à mémoriser}
          { la modification du mode de classement}
          nouveau_mode := obj_mod.modeclassementobjet;
          ancien_mode := OBJ.MODECLASSEMENTOBJET;
          OBJ.MODECLASSEMENTOBJET :=
          obj_mod.modeclassementobjet;
     end;

if obj_mod.motcleobjet1 <> OBJ.MOTCLEOBJET1 then
   begin
        OBJ.MOTCLEOBJET1 :=obj_mod.motcleobjet1;
        ens_attri_clst := ens_attri_clst + [6];
   end;

if obj_mod.motcleobjet2 <> OBJ.MOTCLEOBJET2 then
   begin
        OBJ.MOTCLEOBJET2 :=obj_mod.motcleobjet2;
        ens_attri_clst := ens_attri_clst + [7];
   end;

if  obj_mod.confidentialite <> OBJ.CONFIDENTIALITE
    then
        OBJ.CONFIDENTIALITE := obj_mod.confidentialite;

if obj_mod.motdepasse <> OBJ.MOTDEPASSE then
          OBJ.MOTDEPASSE := obj_mod.motdepasse;

      {on répercute les modifications en BD}
      dbmodify (OBJET,OBJ);
      if dbstatus >= 80 then goto exit;

      {on modifie éventuellement les attributs si}
      {type objet = "document"}
      if obj_mod.typeobjet = 'DOCUMENT' then
        begin
             dbdirect (OBJET, OBJ, ref_objet);
             if dbstatus >= 90 then goto exit;
             dbfpath (DOCU, OBJ, DOO);
             if dbstatus >= 90 then goto exit;

if obj_mod.moderepresentation <> DOCU.MODEREPRESENTATION
               then
DOCU.MODEREPRESENTATION := obj_mod.moderepresentation;

             if obj_mod.typeinfo <> DOCU.TYPEINFO
             then  DOCU.TYPEINFO := obj_mod.typeinfo;
```

```
if obj_mod.referenceoriginal <> DOCU.REFERENCEORIGINAL
                then
DOCU.REFERENCEORIGINAL := obj_mod.referenceoriginal;

             if obj_mod.nomfichier <> DOCU.NOMFICHIER
                then  DOCU.NOMFICHIER := obj_mod.nomfichier;
             if obj_mod.copiede  <> DOCU.COPIEDE
                then  DOCU.COPIEDE := obj_mod.copiede;
             if obj_mod.electronique <> DOCU.ELECTRONIQUE
              then DOCU.ELECTRONIQUE := obj_mod.electronique;

             {on répercute les modifications en BD}
             dbmodify (DOCUMENT, DOCU);
             if dbstatus >= 80 then goto exit;

        end;

      {on reclasse l'objet si sa valeur de l'attribut base }
      {de classement du contenant a été modifiée}

      {on reclasse d'abord dans le contenant d'origine de}
      {l'objet}
      dbfpath (CONTENANTORIGINE, OBJ, -LOD);
      if dbstatus >= 90 then goto exit;
      if dbfound then
        begin
if CONTENANTORIGINE.ATTRIBUTCLASSEMENTOB in ens_attri_clst
        then
            begin
                {si le mode de classement est 0 }
                {on ne fait rien}
                if CONTENANTORIGINE.MODECLASSEMENTOBJET = 1
                   then
                       begin
                            indicateur := 1;
                            trie (CONTENANTORIGINE.xxref,
                            indicateur,code_retour);
                            if code_retour > 0 then goto exit;
                        end;
                 if CONTENANTORIGINE.MODECLASSEMENTOBJET = 2
                    then
                        begin
                              indicateur := 2;
                              trie (CONTENANTORIGINE.xxref,
                                   indicateur, code_retour);
                             if code_retour > 0 then goto exit;
                         end;
              end;
        end;
      {on reclasse ensuite l'objet dans son contenant réel }
      {préalablement on vérifie si lieu d'origine et}
      {lieu réel ne sont confondus}
      dbfpath (CONTENANTREEL, OBJ, -LO);
      if dbstatus >= 90 then goto exit;
```

```
      if dbfound then
          begin
               if not dbequal
               (CONTENANTREEL, CONTENANTORIGINE ) then
                 begin
if CONTENANTREEL.ATTRIBUTCLASSEMENTOB in ens_attri_clst
        then
            begin
                 if CONTENANTREEL.MODECLASSEMENTOBJET = 1
                    then
                        begin
                             indicateur := 1;
                             trie (CONTENANTREEL.xxref,
                             indicateur,code_retour);
                             if code_retour > 0 then goto
                             exit;
                        end;
                  if CONTENANTREEL.MODECLASSEMENTOBJET = 2
                    then
                        begin
                             indicateur := 2;
                             trie (CONTENANTREEL.xxref,
                                indicateur, code_retour);
                           if code_retour > 0 then goto exit;
                        end;
                 end;
        end;

      {si l'attribut base de classement de l'objet }
      {a été modifié :}
      if retrier = true
      {on tient compte du nouveau mode de}
      {de classement éventuel de l'objet}
       then
           begin
                dbdirect (OBJET,OBJ, ref_objet);
                if dbstatus >= 90 then goto exit;
                if OBJ.MODECLASSEMENTOBJET = 1 then
                begin
                     indicateur := 1;
                     trie (ref_objet,
                          indicateur, code_retour);
                     if code_retour > 0 then goto exit;
               end;
               if OBJ.MODECLASSEMENTOBJET = 2 then
               begin
                    indicateur := 2;
                    trie (ref_objet,
                          indicateur, code_retour);
                    if code_retour > 0 then goto exit;
               end;
        end;

        {si l'attribut de base de classement n'a pas}
        {été modifié mais bien le mode de classement}
```

```
if retrier = false then
  if ancien_mode <> nouveau_mode
   then
      begin
       if (ancien_mode = 1) and (nouveau_mode = 2)
        then
          begin
               lieu := 'r';
               inverser_liste (ref_objet,
                               lieu, code_retour);
               if code_retour > 0 then goto exit;
               lieu := 'o';
               inverser_liste (ref_objet,
                               lieu, code_retour);
               if code_retour > 0 then goto exit;
          end;
        if (ancien_mode = 2) and (nouveau_mode = 1)
          then
              begin
                   lieu := 'r';
                   inverser_liste (ref_objet,
                                   lieu, code_retour);
                   if code_retour > 0 then goto exit;
                   lieu := 'o';
                   inverser_liste (ref_objet,
                                    lieu, code_retour);
                   if code_retour > 0 then goto exit;
              end;
         if (ancien_mode = 1 ) and (nouveau_mode = 0)
          then
              begin
              end;
         if (ancien_mode = 2 ) and (nouveau_mode = 0)
          then
              begin
              end;
         if (ancien_mode = 0) and (nouveau_mode = 1)
          then
              begin
                   indicateur := 1; { trie croissant}
                   trie (ref_objet,
                         indicateur, code_retour);
                   if code_retour > 0 then goto exit;
              end;
         if (ancien_mode = 0) and (nouveau_mode = 2)
          then
              begin
                   indicateur := 2;
                   { trie decroissant}
                   trie (ref_objet,
                         indicateur, code_retour);
                   if code_retour > 0 then goto exit;
              end;
end;      end;
```

```
      {on sauve en BD}
      dbcommit;

      code_retour := 0;
      exit: ;

      if dbstatus > 10 then code_retour := dbstatus;
   end;

procedure localisation_origine ;

label exit;

var CONTENANT, CONTENU : TOBJET;

begin
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 3;
                            goto exit;
                       end;
     if CONTENU.TYPEOBJET = 'BUREAU' then
                        begin
                            code_retour := 40;
                            goto exit;
                        end;

     {on recherche le contenant d'origine}
     dbfpath (CONTENANT, CONTENU , -LOD);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 1;
                            goto exit;
                       end;

     {le résultat de la primitive est :}
     ref_contenant := CONTENANT.xxref;

     code_retour := 0;

     exit : ;
     if dbstatus > 10 then
                          begin
                          code_retour := dbstatus;
                          {en cas d'incident ref_contenant}
                          {est mis à null}
                          dbclear (CONTENANT);
                          dbcopyref
                            (CONTENANT.xxref,ref_contenant);
                     end;
```

```
end;

procedure localisation_reelle ;

label exit;

var CONTENANT, CONTENU : TOBJET;

begin
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                        begin
                             code_retour := 3;
                             goto exit;
                        end;
     if CONTENU.TYPEOBJET = 'BUREAU' then
                         begin
                             code_retour := 40;
                             goto exit;
                         end;

     {on recherche le contenant lieu réel}
     dbfpath (CONTENANT, CONTENU , -LO);
     if dbstatus >= 90 then goto exit;
     if not dbfound then
                        begin
                             code_retour := 1;
                             goto exit;
                        end;

     {le résultat de la primitive est :}
     ref_contenant := CONTENANT.xxref;

     code_retour := 0;

     exit : ;
     if dbstatus > 10 then
                      begin
                           code_retour := dbstatus;
                           dbclear (CONTENANT);
                           dbcopyref
                            (CONTENANT.xxref,ref_contenant);
                      end;

end;

procedure changer_lieu_origine ;

label exit;

var ANCIEN : TOBJET;
    CONTENU, CONTENANT : TOBJET;
    INCL : TINCLUSION;
```

```
begin
     dbdirect (OBJET, CONTENU, CONTENU);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                        begin
                             code_retour := 3;
                             goto exit;
                        end;
     if CONTENU.TYPEOBJET = 'BUREAU' then
               begin
                    code_retour := 40;
                    goto exit;
               end;

     {on recherche le nouveau contenant d'origine}
     CONTENANT.xxref := ref_contenant;
     dbdirect (OBJET, CONTENANT, CONTENANT);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then
                         begin
                              code_retour := 31;
                              dbstatus := 0;
                              goto exit;
                         end;
     if not dbfound then
                        begin
                             code_retour := 4;
                             goto exit;
                        end;

     if CONTENANT.TYPEOBJET = 'DOCUMENT' then
                 begin
                       code_retour := 40;
                       goto exit;
                 end;

     { vérification que les références ne désignent}
     {pas le même objet}
     if dbequal (ref_contenant, ref_contenu)
        then
             begin
                  code_retour := 40;
                  goto exit;
             end;

     {verification que l'inclusion est permise }

      INCL.COUPLE :=
         concat (CONTENANT.TYPEOBJET, CONTENU.TYPEOBJET);
      dbid (INCLUSION, INCL);
      if dbstatus >= 90 then goto exit;
```

```
      if not dbfound then
                          begin
                               code_retour := 42;
                               goto exit ;
                          end;

     {enlever de l'ancien lieu d'origine : }

     dbfpath (ANCIEN, CONTENU, - LOD);
     if dbstatus >= 90 then goto exit;
     dbremove (CONTENU, ANCIEN, LOD);
     if dbstatus >= 90 then goto exit;

     {mise a jour de la date de mise a jour de}
     {l'ancien contenant}
     maj_date_maj_contenant (ANCIEN.xxref, code_retour);
     if code_retour > 0 then goto exit;


     { rendre effectif le nouveau lieu d'origine : }

     dbinsert (CONTENU, CONTENANT, LOD);
     if dbstatus >= 80 then goto exit;
     lieu := 'o';
     inserer_nouveau_element (CONTENU.xxref,lieu,
                              CONTENANT.xxref, code_retour);
      if code_retour > 0 then goto exit;

     dbcommit;
     code_retour := 0;
     exit : ;
     if dbstatus > 10 then code_retour := dbstatus;
end;

procedure deplacer_objet ;

label exit;
var ANCIENCONTENANT, CONTENU,NOUVEAUCONTENANT : TOBJET;
    INCL : TINCLUSION;

begin
     {on recherche l'objet à déplacer}
     dbdirect (OBJET, CONTENU, ref_contenu);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                         begin
                              code_retour := 3;
                              goto exit;
                         end;
     if CONTENU.TYPEOBJET = 'BUREAU' then
               begin
                    code_retour := 40;
                    goto exit;
               end;
```

```
{on recherche le nouveua contenant}
dbdirect (OBJET, NOUVEAUCONTENANT, ref_contenant);
if dbstatus >= 90 then goto exit;
if dbstatus = 30 then
                    begin
                         code_retour := 31;
                         dbstatus := 0;
                         goto exit;
                    end;
if not dbfound then
                  begin
                       code_retour := 4;
                       goto exit;
                  end;
if NOUVEAUCONTENANT.TYPEOBJET = 'DOCUMENT' then
            begin
                 code_retour := 40;
                 goto exit;
            end;


{ vérification que les références ne désignent pas}
{le même objet}
if dbequal (ref_contenant, ref_contenu)
   then
        begin
             code_retour := 40;
             goto exit;
        end;

 {verification que l'inclusion est permise }

 INCL.COUPLE :=
 concat (NOUVEAUCONTENANT.TYPEOBJET, CONTENU.TYPEOBJET);
 dbid (INCLUSION, INCL);
 if dbstatus >= 90 then goto exit;
 if not dbfound then
                   begin
                        code_retour := 42;
                        goto exit ;
                   end;
 { retirer le contenu de son ancien contenant }

 dbfpath (ANCIENCONTENANT, CONTENU, - LO);
 if dbstatus >= 90 then goto exit;
 dbremove (CONTENU, ANCIENCONTENANT, LO);
 if dbstatus >= 90 then goto exit;


 {mise a jour de la date de mise a jour de}
 {l'ancien contenant}
 maj_date_maj_contenant
        (ANCIENCONTENANT.xxref, code_retour);
 if code_retour > 0 then goto exit;
```

```
      { inserer le contenu dans le nouveau contenant}
      {en respectant le mode de classement de ce}
      {nouveau contenant }

      lieu := 'r';
      inserer_nouveau_element
         (ref_contenu, lieu, ref_contenant, code_retour);
      if code_retour > 0 then goto exit ;

      dbcommit;
      code_retour := 0;

      exit : ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


procedure reinserer_place_origine  ;

label exit ;

var ANCIENCONTENANT, CONTENU, CONTENANT : TOBJET;

begin
     dbdirect (OBJET, CONTENU, ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                       begin
                            code_retour := 3;
                            goto exit;
                       end;
     if CONTENU.TYPEOBJET = 'BUREAU' then
         begin
              code_retour := 40;
              goto exit ;
         end;
      {test pour voir si l'objet n'est pas déjà à sa}
      {place d'origine au quel cas rien n'est fait :  }

      dbfpath (ANCIENCONTENANT, CONTENU, - LO);
      if dbstatus >= 90 then goto exit;
      dbfpath (CONTENANT, CONTENU, -LOD);
      if dbstatus >= 90 then goto exit;
      if dbequal (ANCIENCONTENANT,CONTENANT) then
                  begin
                        code_retour := 0;
                        goto exit ;
                  end;

     { le contenu est enlevé du contenant actuel : }
     { ce dernier a sa date de maj modifiee    }
```

```
        dbfpath (ANCIENCONTENANT, CONTENU, - LO);
        if dbstatus >= 90 then goto exit;
        dbremove (CONTENU, ANCIENCONTENANT, LO);
        if dbstatus >= 90 then goto exit;


        maj_date_maj_contenant
           (ANCIENCONTENANT.xxref, code_retour);
        if code_retour > 0 then goto exit;


        { recherche du contenant d'origine ; on sait que}
        {dans tous les cas que l'on envisage à ce stade }
        {le contenu est inclus dans un contenant}

        dbfpath (CONTENANT, CONTENU, -LOD);
        if dbstatus >= 90 then goto exit;

        {on insére}
        lieu := 'r';
        inserer_nouveau_element
            (CONTENU.xxref,lieu, CONTENANT.xxref, code_retour);
        if code_retour > 0 then goto exit;

        dbcommit;
        code_retour := 0;
        exit: ;
        if dbstatus > 10 then code_retour := dbstatus;

end;


END.
```

```
UNIT TEXTELIB;

INTERFACE

USES DOS,TYPBUR,DBMS,DECLAR,OUTIL ;

procedure associer_texte_libre ( ref_objet : DBKEY;
                                 nom_fichier : NOMFICHIER;
                                  var code_retour : integer);

procedure detruire_texte_libre (ref_objet : DBKEY ;
                                 var code_retour : integer );

procedure lire_texte_libre (ref_objet : DBKEY;
                                 nom_fichier : NOMFICHIER ;
                                 var code_retour : integer);

IMPLEMENTATION

procedure associer_texte_libre ;

label exit;
type li = string [254];
var OBJ : TOBJET;
    TEXT : TTEXTELIBRE;
    f : file of char;
    ligne : li;
    i,j : integer;
    caractere : char;


begin
     dbdirect (OBJET, OBJ, ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
                      begin
                           code_retour := 3;
                           goto exit;
                      end;

     {on teste s'il existe pas déjà un texte libre}
     dbfpath (TEXT,OBJ,TXT);
     if dbstatus >= 90 then goto exit;
     if dbfound then
                    begin
                         code_retour := 60;
                         goto exit;
                    end;

     {on teste si le fichier est disponible}
     assign (f,nom_fichier);
     {$I-} reset (f) {$I+};
```

```
      if IOresult =  1 then
                        begin
                             close (f);
                             code_retour := 50;
                             goto exit ;
                        end;
      close (f);

      {on procéde à la lecture du fichier et on le stocke}
      {par ligne de 254 caractères}
      reset (f);
      while ( eof (f) = false) do
        begin
             i := 1;
             ligne := '';
             while (i <= 254) and (not eof (f))
               do
                 begin
                      read (f,caractere);
                      ligne  := ligne + caractere;
                      i := i+1;
                 end;

             TEXT.LIGNE := ligne;
             dbcreate (TEXTELIBRE, TEXT);
             if dbstatus >= 80 then goto exit;
             dbinsert (TEXT, OBJ, TXT);
             if dbstatus >= 80 then goto exit;
        end;
      close (f);

      {on détruit le fichier recu pour conserver le principe}
      {d'unicité}
      erase (f);
      dbcommit;
      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
   end;


procedure detruire_texte_libre ;

label exit ;

var  OBJ :TOBJET;
     TEXT : TTEXTELIBRE;

begin
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
```

```
      if not dbfound then
                        begin
                              code_retour := 3;
                              goto exit;
                        end;

      {on teste si le texte libre existe}
      dbfpath (TEXT, OBJ, TXT);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                      begin
                          dbstatus := 61;
                          goto exit;
                      end;
      {on détruit le texte libre stocké en BD}
      while dbfound do
        begin
             dbdelete (TEXTELIBRE, TEXT);
             if dbstatus >= 90 then goto exit;
             dbfpath (TEXT,OBJ, TXT);
             if dbstatus > 90 then goto exit;
        end;

      dbcommit;
      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


procedure lire_texte_libre  ;

label exit;

type li = string [254];

var OBJ : TOBJET;
    TEXT : TTEXTELIBRE;
    f : file of char;
    ligne : li;
    i : integer;

begin
       dbdirect (OBJET, OBJ, ref_objet);
       if dbstatus >= 90 then goto exit;
       if dbstatus = 30 then goto exit;
       if not dbfound then
                        begin
                              code_retour := 3;
                              goto exit;
                        end;
```

```
      {on teste si un fichier avec le même nom n'existe pas }
      {déjà}

      assign (f,nom_fichier);
      {$I-} reset (f) {$I+};
      if IOresult =  1 then
                        begin
                             close (f);
                             code_retour := 50;
                             goto exit ;
                        end;
      close (f);

      {on teste s'il existe un texte libre}
      dbfpath (TEXT, OBJ, TXT);
      if dbstatus >= 90 then goto exit;
      if not dbfound then
                     begin
                           code_retour := 60;
                           goto exit;
                     end;
      rewrite (f);
      while dbfound do
        begin
             for i := 1 to length (TEXT.LIGNE) do
             write (f,TEXT.LIGNE [i]);
             dbnpath (TEXT, OBJ, TXT);
             if dbstatus >= 90 then goto exit;

        end;
      close (f);
      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


END.
```

```
UNIT DOCUMEN;

INTERFACE

USES DOS, TYPBUR, DBMS, DECLAR, OUTIL;

procedure copie (ref_original : DBKEY; nom_objet : NOMOBJET;
            ref_contenant :DBKEY ; nom_fichier: NOMFICHIER;
            ref_copie : DBKEY; var code_retour : integer);

procedure obtenir_corps (ref_objet : DBKEY;
                                  nom_fichier : NOMFICHIER;
                                 var code_retour : integer);

procedure remplacer_corps (ref_objet : DBKEY;
                                nom_fichier : NOMFICHIER;
                                var code_retour : integer);

procedure placer_corps_hors_er (ref_objet : DBKEY;
                                  nom_fichier : NOMFICHIER;
                                 var code_retour : integer);

procedure placer_corps_in_er (ref_objet : DBKEY;
                                  nom_fichier : NOMFICHIER;
                                 var code_retour : integer);

IMPLEMENTATION

procedure copie ;

label exit;
var ORIGINAL, COPIE, CONTENANT ,COPIE2, OBJ: TOBJET;
    DOCU, COPIEDOCU : TDOCUMENT;
    CO, COPIECO : TCORPS;
    TEXT, COPIETEXT : TTEXTELIBRE;
    CAT : TCATEGORIEOBJET;
    INCL : TINCLUSION;
    annee, mois , jour : integer;
    id : string [3];
    code : string[27];

begin
     dbdirect (OBJET, ORIGINAL, ref_original);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
           begin
                code_retour := 3 ;
                goto exit;
           end;

     {on vérifie que le nombre d'occurrences permises }
     {n'est pas dépassé}
```

```
CAT.TYPEOBJETCAT := ORIGINAL.TYPEOBJET;
dbid (CATEGORIEOBJET, CAT);
if dbstatus >= 90 then goto exit;

if CAT.NBOCCUPERMISES < 0
 then
 else
    begin
      CAT.NBOCCUPRESENTES := CAT.NBOCCUPRESENTES + 1;
      if CAT.NBOCCUPRESENTES > CAT.NBOCCUPERMISES
             then
                  begin
                       code_retour := 43;
                       goto exit;
                  end;
    end;

{on vérifie si le contenant existe bien}
dbdirect (OBJET, CONTENANT, ref_contenant);
if dbstatus >= 90 then goto exit;
if dbstatus = 30 then
                     begin
                          dbstatus := 31;
                          goto exit;
                     end;
if not dbfound then
      begin
           code_retour := 4;
           goto exit;
      end;

{on ne fait pas de copie si le type est différent}
{de "document"}
if ORIGINAL.TYPEOBJET <> 'DOCUMENT' then
      begin
            code_retour := 40 ;
            goto exit;
      end;

{on teste si l'inclusion est autorisée dans l'ER}
INCL.COUPLE :=
   concat (CONTENANT.TYPEOBJET, ORIGINAL.TYPEOBJET);
dbid (INCLUSION,INCL);
if dbstatus >= 90 then goto exit;
if not dbfound
    then
         begin
              code_retour := 42;
              goto exit;
         end;

{ on copie les attributs de l'occurrence d'objet }
dbcopyatt (OBJET, ORIGINAL , COPIE);
```

```
  {on teste si l'utilisateur veut donner un code ou}
  {un nom à l'objet}

  if nom_objet = ''
   then
       begin
CAT.DERNIERCODEATTRIBUT := CAT.DERNIERCODEATTRIBUT + 1;
            id := copy (ORIGINAL.TYPEOBJET, 1,3);
            Str (CAT.DERNIERCODEATTRIBUT,code);
            COPIE.NOMOBJET:= Concat (id,'/',code);
       end
   else
       begin
            id := copy (ORIGINAL.TYPEOBJET, 1,3);
            COPIE.NOMOBJET:= Concat (id,'/',nom_objet);
       end;
 dbcreate (OBJET, COPIE);
 if dbstatus >= 80 then goto exit;
 if dbstatus = 2 then
                   begin
                         dbstatus := 0;
                         code_retour := 2;
                         goto exit;
                   end;

 {on copie les attributs précisant le document}
 dbfpath (DOCU, ORIGINAL , DOO);
 if dbstatus >= 90 then goto exit;
 dbcopyatt (DOCUMENT,DOCU, COPIEDOCU);

 {on donne la date de copie }
 date (annee, mois ,jour);
 COPIEDOCU.DATECOPIE.JOURCOPIE := jour;
 COPIEDOCU.DATECOPIE.MOISCOPIE := mois;
 COPIEDOCU.DATECOPIE.ANNEECOPIE := annee;
 COPIEDOCU.NOMFICHIER := nom_fichier;
 COPIEDOCU.COPIEDE := ORIGINAL.NOMOBJET;

 dbcreate (DOCUMENT , COPIEDOCU);
 if dbstatus >= 80 then goto exit;

 {on relie les attributs du document aux attributs}
 {communs (objet)}
 dbinsert (COPIEDOCU, COPIE, DOO);
 if dbstatus >= 80 then goto exit;

 {on copie le corps du docuemnt et on le rattache}
 {à document}
 dbfpath (CO, DOCU, DOC);
 if dbstatus >= 90 then goto exit;
 while dbfound do
      begin
           dbcopyatt (CORPS, CO, COPIECO);
           dbcreate (CORPS, COPIECO);
           if dbstatus >= 80 then goto exit;
```

```
                dbinsert (COPIECO,COPIEDOCU, DOC);
                if dbstatus >= 80 then goto exit;
                dbnpath (CO, DOCU, DOC);
                if dbstatus >= 90 then goto exit;
           end;

      { on copie le texte libre associé à l'objet et}
      {on le rattache à l'occurrence créée}
      dbfpath (TEXT, OBJ, TXT);
      if dbstatus >= 90 then goto exit;
      while dbfound do
           begin
                dbcopyatt (TEXTELIBRE,TEXT, COPIETEXT);
                dbcreate (TEXTELIBRE, COPIETEXT);
                if dbstatus >= 80 then goto exit;
                dbinsert (COPIETEXT, COPIE, TXT);
                if dbstatus >= 80 then goto exit;
                dbnpath (TEXT, OBJ, TXT);
                if dbstatus >= 90 then goto exit;
           end;

      {on donne la valeur de l'identifiant interne de la copie}
      ref_copie := COPIE.xxref;

      {on répercute les modifications dans l'ER}
      dbmodify (CATEGORIEOBJET, CAT);
      if dbstatus >= 80 then goto exit;

      {on classe la copie; le contenant =}
      {le lieu d'origine et le lieu réel}

      lieu := 'r';
      ref_contenu := COPIE.xxref;
      inserer_nouveau_element (ref_contenu,lieu,
                               ref_contenant,code_retour);
      if code_retour > 0 then goto exit;

      lieu := 'o';
      inserer_nouveau_element (ref_contenu,lieu,
                               ref_contenant,code_retour);
      if code_retour > 0 then goto exit;

      {on sauve en bd}
      dbcommit;
      code_retour := 0;

      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;

end;
```

```
procedure obtenir_corps ;

label exit ;
type li = string [254];

var  OBJ :TOBJET;
     DOCU : TDOCUMENT;
     COR : TCORPS;
     f : file of li;
     ligne : li;
begin
     {on recherche l'objet}
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
        begin
             code_retour := 3 ;
             goto exit;
        end;

     if OBJ.TYPEOBJET <> 'DOCUMENT' then
        begin
             code_retour := 40;
             goto exit;
        end;

     {on recherche l'entité document : }
     dbfpath (DOCU, OBJ, DOO);
     if dbstatus >= 90 then goto exit;

     {si l'utilisateur ne fournit pas de nom de}
     { fichier on prend celui donné comme attribut }
     if nom_fichier = '' then
         begin
              nom_fichier := DOCU.NOMFICHIER;
         end;

    {on teste si un fichier avec le même nom }
    {n'existe pas déjà}
    assign (f,nom_fichier);
    {$I-} reset (f) {$I+};
    if IOresult = 0 then
                        begin
                             close (f);
                             code_retour := 51;
                             goto exit ;
                        end;
    close (f);

    {on utilise un outil qui permet de sortir}
    {une copie du corps du document stocké en BD}
    sortie_fichier (nom_fichier,DOCU.xxref,code_retour);
    if code_retour > 0 then goto exit;
```

```
    code_retour := 0;
    exit : ;
    if dbstatus >  10 then code_retour := dbstatus;

end;

procedure remplacer_corps ;

label exit ;

type li = string [254];

var  OBJ :TOBJET;
     DOCU : TDOCUMENT;
     COR : TCORPS;
     f : file of li;
     ligne : li;

begin
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
        begin
             code_retour := 3 ;
             goto exit;
        end;

     if OBJ.TYPEOBJET <> 'DOCUMENT' then
        begin
             code_retour := 40;
             goto exit;
        end;

     {on recherche l'entité document : }
     dbfpath (DOCU, OBJ, DOO);
     if dbstatus >= 90 then goto exit;


     {si l'utilisateur ne fournit pas de nom }
     {de fichier on prend celui donné comme attribut }
     if nom_fichier = '' then
         begin
              nom_fichier := DOCU.NOMFICHIER;
         end;

    {on teste si le fichier est disponible}
    assign (f,nom_fichier);
    {$I-} reset (f) {$I+};
    if IOresult <> 0 then
                        begin
                             close (f);
                             code_retour := 50;
                             goto exit ;
                        end;
```

```
    close (f);

    {on détruit la précédente version du corps du}
    {document stockée en BD que ce soit le corps}
    {proprement dit ou sa référence}
    detruire_corps (ref_objet, code_retour);
    if code_retour > 0 then goto exit;

    {on stocke le nouveau corps du docuement ou sa référence}
    decoupe_fichier (DOCU.xxref, nom_fichier, code_retour);
    if code_retour > 0 then goto exit;

    {on détruit le fichier pour respecter}
    {le principe d'unicité }
    erase (f);

    {on conserve le dernier nom de fichier donné}
    DOCU.NOMFICHIER := nom_fichier;
    dbmodify (DOCUMENT, DOCU);
    if dbstatus >= 80 then goto exit;

    {on met à jour la date de dernière}
    {mise à jour du contenant}
    maj_date_maj_contenant (OBJ.xxref, code_retour);
    if code_retour > 0 then goto exit;

    dbcommit;
    code_retour := 0;
    exit : ;
    if dbstatus > 10 then code_retour := dbstatus;

end;

procedure placer_corps_hors_er ;

label exit ;

type li = string [254];

var  OBJ :TOBJET;
     DOCU : TDOCUMENT;
     COR : TCORPS;
     f : file of li;
     ligne : li;

begin
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
        begin
             code_retour := 3 ;
             goto exit;
        end;
```

```
 if OBJ.TYPEOBJET <> 'DOCUMENT' then
    begin
         code_retour := 40;
         goto exit;
    end;

 {on recherche l'entité document : }
 dbfpath (DOCU, OBJ, DOO);
 if dbstatus >= 90 then goto exit;

 {si l'utilisateur ne fournit pas de nom de}
 {fichier on prend celui donné comme attribut }
 if nom_fichier = '' then
     begin
          nom_fichier := DOCU.NOMFICHIER;
     end;

{on teste si le corps du document est bien stocké en BD}
if DOCU.STOCKAGE = false then
     begin
          code_retour := 40;
          goto exit;
     end;

{on teste si un fichier avec le même nom }
{n'existe pas déjà}
assign (f,nom_fichier);
{$I-} reset (f) {$I+};
if IOresult = 0 then
                    begin
                         close (f);
                         code_retour := 51;
                         goto exit ;
                    end;
close (f);

{on utilise la primitive qui permet d'obtenir}
{le corps du document}
obtenir_corps (ref_objet, nom_fichier, code_retour);
if code_retour > 0 then goto exit;

{on détruit le corps du document dans la BD}
detruire_corps (ref_objet, code_retour);
if code_retour > 0 then goto exit;

{on modifie l'attribut stockage :}
DOCU.STOCKAGE := false;
dbmodify (DOCUMENT, DOCU);
if dbstatus  >= 80 then goto exit;

{le corps du document est maintenant}
{la référence à ce corps càd le nom du fichier sous}
{lequel il est placé hors de l'ER}
```

```
    COR.LIGNEDOC := nom_fichier;
    dbcreate (CORPS, COR);
    if dbstatus >= 80 then goto exit;

    {on rattache le corps à l'entité document}
    dbinsert (DOCU,COR,DOC);
    if dbstatus >= 80 then goto exit;

    dbcommit;
    code_retour := 0;
    exit : ;
    if dbstatus > 10 then code_retour := dbstatus;

end;

procedure placer_corps_in_er ;

label exit ;

type li = string [254];

var  OBJ :TOBJET;
     DOCU : TDOCUMENT;
     COR : TCORPS;
     f : file of li;
     ligne : li;

begin
     dbdirect (OBJET, OBJ ,ref_objet);
     if dbstatus >= 90 then goto exit;
     if dbstatus = 30 then goto exit;
     if not dbfound then
        begin
             code_retour := 3 ;
             goto exit;
        end;

     if OBJ.TYPEOBJET <> 'DOCUMENT' then
        begin
             code_retour := 40;
             goto exit;
        end;

       {on recherche l'entité document : }
     dbfpath (DOCU, OBJ, DOO);
     if dbstatus >= 90 then goto exit;

     {si l'utilisateur ne fournit pas de nom}
     {de fichier on prend celui donné comme attribut }
     if nom_fichier = '' then
         begin
              nom_fichier := DOCU.NOMFICHIER;
         end;
```

```
     {on teste si le corps figure déjà dans la BD}
     if DOCU.STOCKAGE = true then
          begin
               code_retour := 40;
               goto exit;
          end;

     {on teste si le fichier est disponible}
     assign (f,nom_fichier);
     {$I-} reset (f) {$I+};
     if IOresult <> 0 then
                         begin
                              close (f);
                              code_retour := 50;
                              goto exit ;
                         end;
     close (f);

     {on utilise un outil qui stocke un fichier dans}
     {la BD et le rattache au document}
     decoupe_fichier (DOCU.xxref, nom_fichier, code_retour);
     if code_retour > 0 then goto exit;

     {on efface le fichier recu afin de conserver}
     {le principe d'unicité de l'information}
     erase (f);

     {on modifie l'attribut qui permet de savoir }
     {si le corps du document}
     {est en BD}
     DOCU.STOCKAGE := true;

     {on conserve le dernier de fichier recu}
     DOCU.NOMFICHIER := nom_fichier;
     dbmodify (DOCUMENT, DOCU);
     if dbstatus >= 80  then goto exit;

     dbcommit;
     code_retour := 0;
     exit : ;
     if dbstatus > 10 then code_retour := dbstatus;

end;

END.
```

```
UNIT IDENTIF;

INTERFACE

USES DOS, TYPBUR,DBMS;

procedure mettre_a_null
   (var ref_objet : DBKEY; var code_retour : integer);

function verif_ident (id_objet : DBKEY):boolean;

IMPLEMENTATION

procedure mettre_a_null;

label exit;

var OBJ : TOBJET;

begin
      dbdirect  (OBJET,OBJ,ref_objet);
      if dbstatus >= 90 then goto exit;
      dbclear (OBJ);
      ref_objet := OBJ.xxref;
      code_retour := 0;
      exit: ;
      if dbstatus > 10 then code_retour := dbstatus;
end;


function verif_ident ;

var OBJ : TOBJET;


begin
     if dbnull (id_objet) then verif_ident := true
                          else verif_ident := false;
end;

END.
```

7. Les macro primitives

```
UNIT macro;


INTERFACE


USES DOS, CRT, TYPBUR, DBMS, DECLAR, OUTIL, ACCES, CREAMAJ,
     TEXTLIB, DOCUMEN, declarmp;

procedure feuilleter_objet (ref_objet : DBKEY;
                        var debut_liste : pointeur_chaine;
                        var status_mp : integer);

procedure vider_objet (ref_objet : DBKEY; niveau : integer;
                        var status_mp : integer);

procedure modifier_commentaire(ref_objet: DBKEY;
                        nom_fichier : nomfichier;
                        var status_mp : integer);

procedure mettre_ordre (ref_objet : DBKEY;
                        var status_mp : integer);



IMPLEMENTATION


procedure const_chainon (ref_composant : dbkey;
                         indiloca : integer;
                         var fin_liste : pointeur_chaine);

var inter_liste : pointeur_chaine;

begin
  new(inter_liste);
  inter_liste^.ref := ref_composant;
  inter_liste^.indic := indiloca;
  inter_liste^.suivant := nil;
  fin_liste^.suivant := inter_liste;
  fin_liste := inter_liste;
end;
```

```
procedure feuilleter_objet;

label sortie;

var
  fin_liste : pointeur_chaine;
  ref_composant : DBKEY;
  indiloca, code_retour : integer;

begin
  status_mp := 0;
  new(debut_liste); debut_liste :=  nil;
  new(fin_liste); fin_liste :=  nil;
  acces_premier (ref_objet, ref_composant, indiloca,
  code_retour); if code_retour > 0
    then
      begin
        status_mp := code_retour;
        if code_retour = 4  then status_mp := 3;
        if code_retour = 31 then status_mp := 30;
        goto sortie
      end
    else
      begin
        debut_liste^.ref     := ref_composant;
        debut_liste^.indic   := indiloca;
        debut_liste^.suivant := nil;
        fin_liste := debut_liste;

        while code_retour = 0 do
          begin
            acces_suivant (ref_objet, ref_composant,
            indiloca, code_retour); if code_retour = 0
              then const_chainon (ref_composant, indiloca,
                                  fin_liste)
              else
                begin
                  if code_retour = 1 then status_mp := 0
                                     else status_mp :=
                  code_retour; goto sortie
                end
          end;
      end;
sortie : ;
end;
```

```
procedure vider_objet (ref_objet : DBKEY; niveau : integer;
                       var status_mp : integer);
label sortie;

var
  debut                  : pointeur_chaine;
  attributs              : objet_attr;
  retour,status, stat    : integer;

begin
  status_mp := 0;
  feuilleter_objet (ref_objet, debut, status);
  if status > 1
    then begin status_mp := status; goto sortie; end
    else
      begin
        if status = 0
          then
            repeat
              caracteristiques(debut^.ref, attributs, stat);
              if stat = 0
                then
                  begin
                    if attributs.typeobjet = 'DOCUMENT'
                      then
                        begin
                          if ((niveau = 3) or (niveau =
                                               debut^.indic))
                            then detruire_objet
                            (debut^.ref,retour);
                          if retour > 0
                            then
                              begin
                                status_mp := retour;
                                goto sortie;
                              end;
                        end
                      else vider_objet(debut^.ref, niveau,
                                       status);
                    debut := debut^.suivant;
                  end
                else begin
                       status_mp := stat; goto sortie; end
            until debut = nil;
      end;
  sortie : ;
end;

procedure modifier_commentaire;
begin
  status_mp := 0;
  detruire_texte_libre (ref_objet, status_mp);
  if status_mp = 0
    then associer_texte_libre (ref_objet, nom_fichier,
                               status_mp); end;
```

```
procedure mettre_ordre;

label sortie;

var
  debut  : pointeur_chaine;
  status : integer;

begin
  status_mp := 0;
  feuilleter_objet (ref_objet, debut, status);
  if status > 1 then status_mp := status;
  if status = 0
    then
      repeat
        if (debut^.indic = 1) then
        reinserer_place_origine(debut^.ref,status);
        if status > 0 then
          begin
            status_mp := status;
            goto sortie;
          end
                      else debut := debut^.suivant;
      until (debut = nil);
  sortie : ;
end;

end.
```

8. Le code source de l'exemple d'application

```
program DEMO;

USES DOS, CRT, TYPBUR, DBMS, DECLAR, OUTIL, INITIALI,
   STRUCTUR, ACCES, CREAMAJ, TEXTLIB, DOCUMEN, IDENTIFI,
   DECLARMP, MACRO;

label sortie;

const ligneb = '                                          ';

var
  result1 : DBKEY;
  choix : char;
  cd : integer;
  fin, result2 : boolean;


procedure tableau(entreeref : DBKEY;
                  var sortieref : DBKEY;
                  var quitter : boolean); label fin;


var

  debut, parcours : pointeur_chaine;
  attributs : objet_attr;
  choix, i, j, st, cdret : integer;
  ch : char;

begin
  quitter := false;
  clrscr;
  caracteristiques(entreeref, attributs, st);
  if st > 0 then begin quitter := true; goto fin; end;
  writeln('');
  write('Etat de l objet (de type ',attributs.typeobjet);
  writeln(') appelé ',attributs.nomobjet);
  writeln(''); writeln('');
  if (attributs.typeobjet = 'DOCUMENT')
   then begin
         repeat
          gotoxy(3,14);
          write('Sélection de cet objet (S) ou Quitter (Q)');
          read(ch)
         until ch in ['S','Q'];
         case ch of
           'S' : sortieref := entreeref;
           'Q' : quitter := true;
         end;
         goto fin;
        end;
```

```
feuilleter_objet(entreeref, debut, cdret);
parcours := debut;
if cdret > 1 then begin quitter := true; goto fin; end;
if cdret = 1
  then
    begin
      repeat
        gotoxy(3,12); write('Cet objet de bureau est
        vide'); gotoxy(3,19); write(ligneb, ligneb);
        gotoxy(3,19); write('Sélection de cet objet (S) ou
                             Quitter (Q) '); read(ch)
      until ch in ['S','Q'];
      case ch of
        'S' : sortieref := entreeref;
        'Q' : quitter := true;
      end;
      goto fin;
    end
  else
   begin
    i := 0;
     repeat
      caracteristiques(parcours^.ref, attributs, st);
      if st > 0 then goto fin;
      i := i + 1;
write('(',i,') ','Composant de type',attributs.typeobjet);
writeln(', appelé ',attributs.nomobjet);
      parcours := parcours^.suivant;
     until (parcours = nil);
   end;
repeat
  gotoxy(3,24); write(ligneb, ligneb);
  gotoxy(3,24);
  write('Choisir un élément de ce tableau (C) ou Quitter
  (Q) '); read(ch);
until ch in ['C','Q'];
case ch of
  'C' : begin
          gotoxy(3,24); write(ligneb, ligneb);
          gotoxy(3,24); write('Quel élément choisir dans le
          tableau'); write(' (de 1 à ',i,') : ');
          read(choix);
          repeat
            gotoxy(3,24); write(ligneb, ligneb);
            gotoxy(3,24); write('Sélectionner (S) ou Etat
            (E) de cet objet (',choix,')'); read(ch)
          until ch in ['S','E'];
          case ch of
            'S' : begin
                    parcours := debut;
                    for j := 1 to (choix-1) do
                      parcours := parcours^.suivant;
                    sortieref := parcours^.ref;
                  end;
```

```
              'E' : begin
                      parcours := debut;
                      for j := 1 to (choix-1) do
                        parcours := parcours^.suivant;
                      tableau(parcours^.ref, sortieref,
                              quitter);
                    end;
            end;
          end;
    'Q' : quitter := true;
  end;

fin: ;
end;



procedure etater(entree : DBKEY);

var
  resultat : DBKEY;
  quitter : boolean;
  toto : char;

begin
  gotoxy(20,10);
  write('Attention : le choix "Sélection" est actif mais
  donne ');
  gotoxy(20,11);write  ('            le même résultat que
  "Quitter" ? '); gotoxy(20,14);
  write('Enfoncez une touche pour continuer ');
  repeat until keypressed;
  tableau(entree, resultat, quitter);
end;

procedure mise_a_blanc(var ensemble_attributs : objet_attr);
begin
  with ensemble_attributs do
    begin
      nomobjet := ''; typeobjet := '';
      createurresponsable := ''; modeclassementobjet := 0;
      attributclassementob := 1; motcleobjet1 := '';
      motcleobjet2 := 0;
    end;
end;
```

```
procedure creerobjet(result1 : dbkey;
                     var resultat : boolean);

var
  liste_caract : objet_attr;
  recom, rech, stock : char;
  futur : string[30];
  cr : integer;
  retour, retfutur : DBKEY;
  indic : boolean;

begin
  repeat
    clrscr;
    mise_a_blanc(liste_caract);

    with liste_caract do
      begin
        resultat := true;
        writeln; writeln;
        write('          Liste des attributs de l objet à
        créer '); writeln; writeln; writeln;
        write('   Nom de l objet à créer  : ');
        readln(nomobjet); readln(nomobjet);
        write('   Type de l objet à créer : ');
        readln(typeobjet); write(' Créateur de l objet : ');
        readln(createurresponsable);
        if typeobjet <> 'DOCUMENT' then
          begin
            writeln('   Mode de classement (0) non trie, (1)
            tri croissant,');
            write  (' (2) tri décroissant : ');
            readln(modeclassementobjet);
            write('   Attribut à la base du classement : ');
            read(attributclassementob);
          end;
        write('   Premier mot clé (carac) : ');
        readln(motcleobjet1); readln(motcleobjet1);
        write('   Deuxième mot clé (nombre) : ');
        readln(motcleobjet2 ); confidentialite:=FALSE;

        if typeobjet = 'DOCUMENT' then
          begin
            writeln; writeln;
            write (' Mode de représentation du document : ');
            readln(moderepresentation) ;
            write ('   Type de l information  : ');
            readln(typeinfo);
            write ('   Référence à l original : ');
            readln(referenceoriginal);
            write ('   Nom du fichier         : ');
            readln(nomfichier);
```

```
      write ('   Stockage dans l ER (O/N)       : ' );
      readln (stock);
      if stock = 'O' then stockage := true else
      stockage := false;
    end ;
end;

writeln; writeln; writeln; writeln;
write('Pressez une touche pour continuer l opération de
      création  ');
repeat until keypressed;

repeat
  clrscr;
  writeln; writeln; writeln;
  writeln('Recherche du lieu de rangement du nouvel
          objet de bureau ');
  writeln; writeln; writeln; writeln;

  repeat
    gotoxy(3,6); writeln(ligneb, ligneb);
    writeln(ligneb, ligneb); writeln(ligneb);
    gotoxy(3,6);
    writeln('Recherche de ce lieu en parcourant le
             bureau (P)');
    write  ('  ou accès direct à cet objet par son nom
              (A) : ');
    read(rech);
  until rech in ['P','A'];

  cr := 99; indic := true;

  case rech of
    'P' : tableau(result1, retfutur, indic);
    'A' : begin
            gotoxy(3,10); writeln;writeln(ligneb,
            ligneb); writeln(ligneb, ligneb);
            gotoxy(3,10); writeln;
            write('Nom du futur lieu de rangement : ');
            readln(futur);readln(futur);
            acces_direct(futur,retfutur, cr);
          end;
  end;
 until ((cr = 0) or (indic = false));

 creer_objet(liste_caract, retfutur, retour, cr);

 case cr of
   0 : begin
         writeln('La création est faite ');
         writeln('(enfoncer une touche pour retourner
         au menu principal)'); repeat until keypressed;
       end;
```

```
2 : begin
      writeln('Un objet de bureau avec ce nom existe
               déjà');
      write('Voulez-vous recommencer l opération de
             création (O/N)');
      readln(recom); if recom = 'N' then cr := 0;
    end;

40 : begin
       writeln('Opération de création interdite pour
                ce type d objet');
       write('Voulez-vous recommencer l opération de
              création (O/N)');
       readln(recom);
       if recom = 'N' then cr := 0;
     end;

42 : begin
       writeln('Inclusion interdite par la structure
                de l ER');
       write('Voulez-vous recommencer l opération de
              création (O/N)');
       readln(recom);
       if recom = 'N' then cr := 0;
     end;

43 : begin
       writeln('Dépassement du nombre maximum d
                occurrences de ce type d objet');
       write('Voulez-vous recommencer l opération de
              création (O/N)');
       readln(recom);
       if recom = 'N' then cr := 0;
     end;

50 : begin
       write('Le fichier identifié par
              (',liste_caract.nomfichier);
       writeln(') n a pas été trouvé');
       write('Voulez-vous recommencer l opération de
              création (O/N)');
       readln(recom);
       if recom = 'N' then cr := 0;
     end;

80 : begin
       write('Plus de place dans le bureau pour ce
              objet');
       write('Voulez-vous recommencer l opération de
              création (O/N)');
       readln(recom);
       if recom = 'N' then cr := 0;
     end;
```

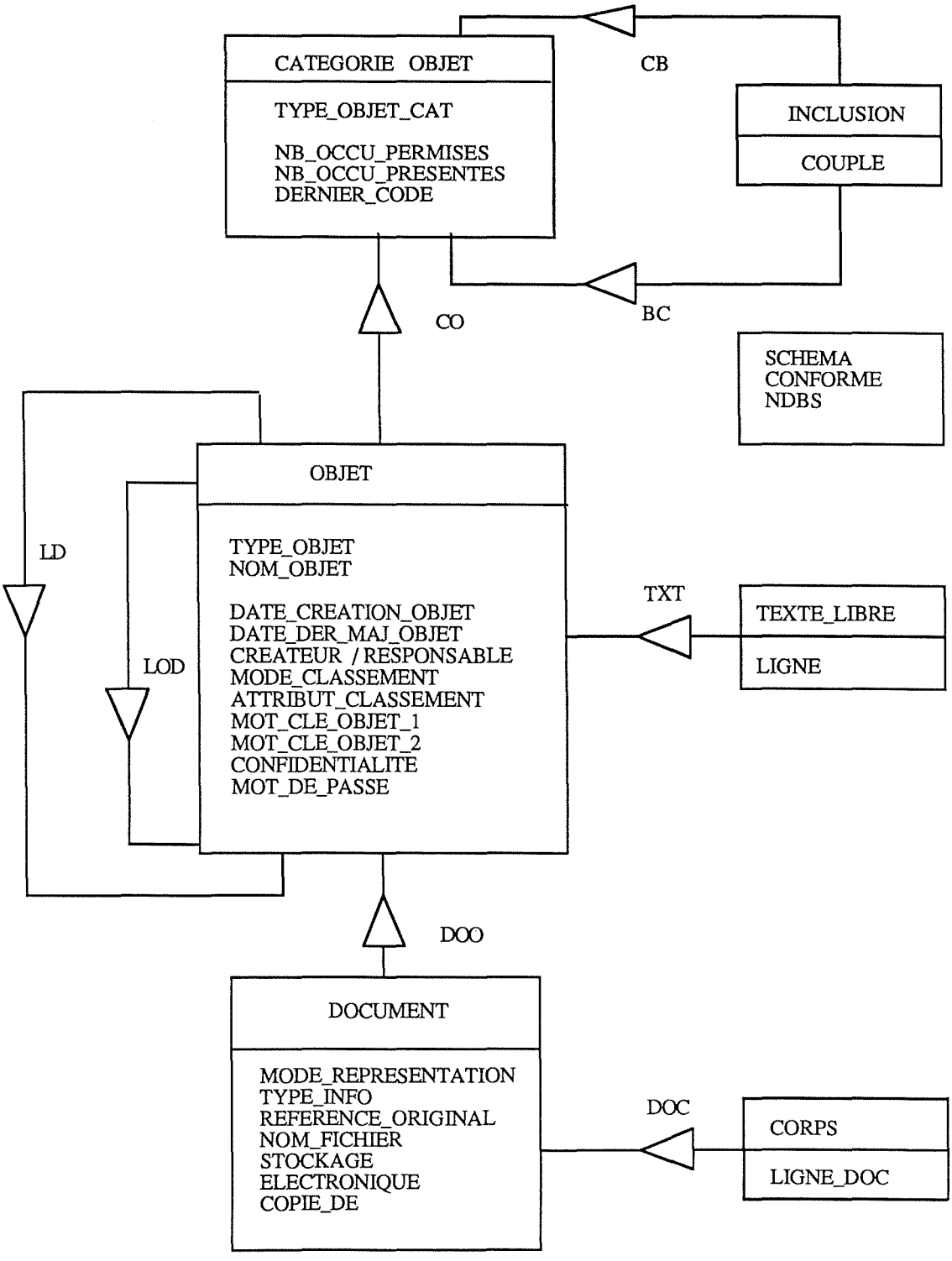
```
        else begin
               write('Problèmes techniques lors de l
                     opération');
               write('Voulez-vous recommencer l opération de
                     création (O/N)');
               readln(recom);
               if recom = 'N' then cr := 0;
             end;
      end;
  until (cr = 0);
end;


procedure information_objet(result1 : dbkey);

label sortie;

var
  retfutur : DBKEY;
  indic : boolean;
  cr : integer;
  cara : objet_attr;
  retour, rech : char;
  futur : string[30];

begin
    writeln; writeln; writeln;
    writeln('Informations sur quel objet de bureau ');
    writeln; writeln; writeln; writeln; writeln;

    repeat
      repeat
        gotoxy(3,10); writeln(ligneb, ligneb);
        writeln(ligneb, ligneb); writeln(ligneb);
        gotoxy(3,12); writeln;writeln(ligneb, ligneb);
        writeln(ligneb, ligneb); gotoxy(3,10);
        writeln('Recherche de l objet en parcourant le
                 bureau (P)');
        write  ('  ou accès direct à cet objet par son
                   nom (A) : ');
        read(rech);
      until rech in ['P','A'];

      case rech of
        'P' : tableau(result1, retfutur, indic);
        'A' : begin
                gotoxy(3,12); writeln; writeln(ligneb,
                ligneb); writeln(ligneb, ligneb);
                gotoxy(3,12); writeln;
                write('  Nom de l objet : ');
                readln(futur);readln(futur);
                acces_direct(futur,retfutur, cr);
              end;
         end;
    until ((cr = 0) or (indic = false));
```

```
caracteristiques(retfutur, cara, cr);
if cr > 0 then goto sortie;

clrscr; writeln; writeln;
writeln('Informations sur les objets de bureau ');
writeln; writeln; writeln;
  with cara do
    begin
     writeln('Nom de l objet  : ',nomobjet);
     writeln('Type de l objet : ',typeobjet);
     write('Date de création :
           ',datecreationobjet.jour,'/');

     writeln(datecreationobjet.mois,'/',
             datecreationobje t.annee);
     write('Date de dernière mise à jour :
           ',datedermaj.jourmaj,'/');

     writeln(datedermaj.moismaj,'/',datedermaj.anneemaj);
     write('Nom du responsable ou du créateur de
            l objet : ');
     writeln(createurresponsable);
     if typeobjet <> 'DOCUMENT'
       then
        begin
          write('Mode de classement de l objet : ');
          case modeclassementobjet of
            0 : writeln('non trié');
            1 : writeln('trié croissant');
            2 : writeln('trié décroissant');
          end;
          write('Attribut à la base du classement : ');
            writeln(attributclassementob);
        end;
      writeln('Premier mot clé : ',motcleobjet1);
      writeln('Deuxième mot clé : ',motcleobjet2);
      If confidentialite
        then
         begin
           writeln('Objet confidentiel');
           writeln('Mot de passe : ',motdepasse);
         end
        else writeln('Objet non confidentiel');
      If typeobjet = 'DOCUMENT'
        then
          begin
            writeln('Mode de représentation :
                    ',moderepresentation);
            writeln('Type de document : ',typeinfo);
            writeln('Référence à l original :
                    ',referenceoriginal);
```

```
                writeln('Copie du document : ',copiede);
                write('Faite le : ',datecopie.jourcopie,'/');
writeln(datecopie.moiscopie,'/',datecopie.anneecopie);
              end;
        end;
  repeat
    gotoxy(3,24); write(ligneb, ligneb);
    gotoxy(3,24); write('Retour au menu principal (R) ');
    readln(retour);
  until retour in ['R'];
  sortie :;
end; {fin de la procedure information_objet}



procedure bougerreel(entreeboug : DBKEY);

label sortieboug;

var
  choixmenu, choixboug, choixboug1 : char;
  objetboug, nvrangement : DBKEY;
  idboug, idboug1 : boolean;
  bougcr : integer;

begin
  writeln(' '); writeln(' ');
  writeln('Déplacer un objet de bureau');
  writeln(' '); writeln(' ');
  writeln('Recherche de l objet à déplacer en parcourant le
           bureau ');
  writeln('(introduisez un caractère pour commencer la
            recherche');
  readln(choixboug); tableau(entreeboug, objetboug, idboug);
  if (idboug = true) then goto sortieboug;

  clrscr;
  writeln(' '); writeln(' ');
  writeln('Déplacer un objet de bureau');
  writeln(' '); writeln(' ');
  writeln('Recherche du nouveau lieu de rangement ');
  writeln('(introduisez un caractère pour commencer la
           recherche');
  readln(choixboug1);
  tableau(entreeboug, nvrangement, idboug1);
  if (idboug1 = true) then goto sortieboug;

  deplacer_objet(objetboug, nvrangement, bougcr);
  if bougcr > 0
    then
      writeln('Problèmes lors du déplacement, abandon de l
               opération')
```

```
    else
      writeln('L objet a été déplacé ');
  writeln('Retour au menu principal (entrez un caractère
          quelconque) ');
  readln(choixmenu);

sortieboug:;
end;{fin de la procedure bougerreel}



procedure detruire(entreedetr : dbkey);

label sortiedetr;

var
  choixdet, choixmenu : char;
  objdetr : DBKEY;
  inddtr : boolean;
  nomdtr : string[30];
  detrcr, addtr : integer;

begin
  inddtr := false;
  writeln(' '); writeln(' ');
  writeln('Destruction d un objet de bureau');
  writeln(' '); writeln(' ');
  writeln('Recherche de l objet à détruire en parcourant le
          bureau ');
  writeln('(introduisez un caractère pour commencer la
           recherche');
  readln(choixdet);
  tableau(entreedetr, objdetr, inddtr);
  if (inddtr = true) then goto sortiedetr;

  detruire_objet(objdetr, detrcr);
  if detrcr > 0
    then
      writeln('Problèmes lors de la destruction, abandon de l
               opération')
    else
      writeln('L objet a été détruit ');

      writeln('Retour au menu principal (entrez un caractère
           quelconque) ');
      readln(choixmenu);
    end;
sortiedetr:;

end;{fin de la procedure détruire}
```

```
{corps du programme}
begin

  fin := false;
  ouvrir_er('BUREAU', cd);
  if cd > 0 then goto sortie;
  write('ouverture ok');
  donner_premier('BUREAU',result1, cd);
  if cd > 0 then goto sortie;
  write('donner_premier ok');

Repeat

 clrscr; writeln(''); writeln('');
 writeln('                          - MENU PRINCIPAL -');
 writeln(''); writeln(''); writeln('');
 writeln('   - 1 -  Etat de l environnement de rangement');
 writeln(''); writeln('  - 2 - Creer un objet de bureau');
 writeln(''); writeln('  - 3 - Information sur un objet de
         1 environnement de rangement'); writeln('');
 writeln(' -4 -Déplacer un objet de bureau (lieu réel) ');
 writeln(''); writeln(' - 5 - Détruire un objet de bureau');
 writeln(''); writeln(' - 6 -Sortie');
 writeln(''); writeln('');

  Repeat
    gotoxy(3,21);
    write('   Votre choix S.V.P. (1-6) ' );
    gotoxy(32,21); write('           ');gotoxy(32,21);
    read(choix);
  Until (choix in ['1','2','3','4','5','6']);

  Case choix of
    '1' : begin clrscr; ETATER(result1); end;
    '2' : begin clrscr; CREEROBJET(result1, result2); end;
    '3' : begin clrscr; INFORMATION_OBJET(result1); end;
    '4' : begin clrscr; BOUGERREEL(result1) ;end;
    '5' : begin clrscr; DETRUIRE(result1); end;
    '6' : begin clrscr; fin := true; end;
  end;
Until (fin = true);
sortie:;

fermer_er('BUREAU', CD);
end.
```


CATEGORIE OBJET
TYPE_OBJET_CAT
NB_OCCU_PERMISES
NB_OCCU_PRESENTES
DERNIER_CODE
CB
INCLUSION
COUPLE
CO
BC
SCHEMA
CONFORME
NDBS
OBJET
TYPE_OBJET
NOM_OBJET
DATE_CREATION_OBJET
DATE_DER_MAJ_OBJET
CREATEUR / RESPONSABLE
MODE_CLASSEMENT
ATTRIBUT_CLASSEMENT
MOT_CLE_OBJET_1
MOT_CLE_OBJET_2
CONFIDENTIALITE
MOT_DE_PASSE
LD
LOD
TXT
TEXTE_LIBRE
LIGNE
DOO
DOCUMENT
MODE_REPRESENTATION
TYPE_INFO
REFERENCE_ORIGINAL
NOM_FICHIER
STOCKAGE
ELECTRONIQUE
COPIE_DE
DOC
CORPS
LIGNE_DOC